Evolution einer 3D-Beschreibung aus Bildern mit Hilfe von Lindenmayer-Systemen

- Diplomarbeit -

zur Erlangung des akademischen Grades

Diplom-Informatiker (FH)

im

Fachbereich Informatik und Medien

der Fachhochschule Brandenburg

vorgelegt von

Jan Derer

14. Juni 2004

1. Gutachter : Prof. Dr. F. Mündemann

2. Gutachter : Dipl.-Inf. I. Boersch

Erklärung

Ich erkläre hiermit, dass die vorliegende Arbeit von mir selbst und ohne fremde Hilfe verfasst wurde. Alle benutzten Quellen sind im Literaturverzeichnis angegeben. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Brandenburg, 14. Juni 2004

Jan Derer

Kurzfassung

Diese Arbeit beschäftigt sich mit den Lindenmayer-Systemen (kurz L-Systemen), deren Evolution und Anwendungsmöglichkeiten.
Dazu werden die theoretischen Grundlagen zu den L-Systemen und ihre wichtigsten Erweiterungen dargestellt. Des Weiteren werden einige Klassifikationen von L-Systemen präsentiert. Außerdem wird auf die graphische Interpretation des Wortes eines L-Systems mit Hilfe der Turtle-Interpretation eingegangen.
Für den Einstieg in die Evolution wird ein kurzer Überblick zu den Genetischen Algorithmen und zur Genetischen Programmierung gegeben. Im Anschluss werden die Mutationskonzepte besprochen und Anwendungsmöglichkeiten präsentiert.
Aus praktischer Sicht wird eine Applikation vorgestellt, die als Eingabe ein L-System bekommt und eine 3D-Grafik als Ausgabe liefert. Dabei ist die Applikation speziell auf die Bedürfnisse der Evolution zugeschnitten. Zusätzlich wird eine graphische Oberfläche für die Applikation entwickelt, um 3D-Grafiken etwas komfortabler aus L-Systemen erzeugen zu können.

Abstract

This thesis is part of an evolving bigger thesis. It will be concerned with Lindenmayer-Systems (or L-Systems) and their evolution. The most theoretical basics of L-Systems will be shown as well as important additions to the basic L-System. From a theoretical perspective, some classifications and an interpretation of L-Systems to generate a 3d-graphic are explained. As a basis for the following thesis about the evolution of L-Systems, some concepts of mutation and possible applications will be discussed. .
From a practical point of view, an application will be presented, that gets a L-System as input and provides a 3d-graphic as output. The program is especially designed for evolutionary loops of L-Systems. Additionally to this commandline-based application, a GUI has been developed to generate the 3d-graphic of the L-System in a comfortable way.

Keywords: L-Systems, simulated evolution, mutation, applications of L-Systems, Lparser, POV-Ray

Inhaltsverzeichnis

1 EINLEITUNG ZU DIESER DIPLOMARBEIT 1

1.1 Motivation zur Modellierung von Pflanzen mit L-Systemen 1

1.2 Aufgabenstellung 3

1.3 Gliederung 3

1.4 Anforderungen an den Leser 4

2 THEORETISCHE GRUNDLAGEN VON L-SYSTEMEN UND DEREN EVOLUTION 5

2.1 Kurzbiografien von wichtigen Personen 5

2.2 L-Systeme 6
2.2.1 Wichtige Unterschiede zu den Chomskysprachen 6
2.2.2 Das ursprüngliche L-System 7
2.2.3 Übersicht der L-Systeme 9
2.2.3.1 Basis L-System 10
2.2.3.2 Deterministisches L-System 12
2.2.3.3 Propagierendes L-System 12
2.2.3.4 L-Systeme mit Terminalzeichen 14
2.2.3.5 L-Systeme mit Tabellen 16
2.2.3.6 L-Systeme mit Verzweigungen 18
2.2.3.7 Stochastische L-Systeme 20
2.2.3.8 Parametrisierte L-Systeme 21
2.2.3.9 Kontextsensitive L-Systeme 22
2.2.3.10 Weitere L-Systeme 23
2.2.3.11 Kombination von L-Systemen 24
2.2.4 Wachstumsfunktion 25
2.2.5 Klassifikation / Einordnung 26
2.2.6 Interpretation des Wortes 28
2.2.7 Übersicht über Anwendungsmöglichkeiten von L-Systemen 34

2.3 Alternativen zu L-Systemen 36
2.3.1 P-Systeme 37
2.3.2 Eco-Grammar Systeme 37

2.4 Genetische Algorithmen 39

2.5 Genetische Programmierung 41

2.6 Mutation von L-Systemen 46
2.6.1 Mutation von L-Systemen unter Genetischen Algorithmen 46
2.6.2 Mutation von L-Systemen unter Genetischer Programmierung 54
2.6.3 Mutation der Interpretation 56

2.7 Applikationsvision 56
2.7.1 Das Grundgerüst 58
2.7.2 Die Anwendungsmöglichkeiten 58

3 PROGRAMME, DIE L-SYSTEME SIMULIEREN/INTERPRETIEREN 61

3.1 Lparser 61

3.2 L-System 62

3.3 RayTraced Evolution 63

3.4 LinSys3D 64

3.5 Zusammenfassung und Übersicht 66

4 ENTWURF UND IMPLEMENTIERUNG DER FASSADE 69

4.1 Der Begriff der Fassade 69

4.2 Einführung zum Entwurf 69

4.3 Die Entwicklungsumgebung 70

4.4 Modulübersicht 70

4.5 Vereinfachungen und die entsprechende Modulübersicht 72

4.6 Lparser 75

4.7 Persistence of Vision-Ray 77

4.8 Schnittstelle der Fassade 80

4.9 Ablauf der Fassade 82

4.10 Implementierung der Fassade 86

5 DIE GRAPHISCHE OBERFLÄCHE 91

5.1 Einführung zur graphischen Oberfläche 91

5.2 Die Entwicklungsumgebung für die graphische Oberfläche 91

5.3 Externe Bibliotheken und Komponenten 91

5.4 Entwurf der graphischen Oberfläche 92
5.4.1 Das Hauptformular - TMDIMain 92
5.4.2 Das Kindfenster - TMDIChildMain 94
5.4.3 Einstellungs-Dialog - TCFGForm 95
5.4.4 Die 3D-Ansicht - TDirect3DForm 96

6 TESTS UND AUSWERTUNG DER ENTWICKELTEN SOFTWARE 98

6.1 Test der Fassade 98
6.1.1 Testumgebung 98
6.1.2 Testumfang 98
6.1.3 Testdurchführung 98
6.1.4 Auswertung 99
6.1.4.1 Performancetest 99
6.1.4.2 Belastungstest 101
6.1.4.3 Funktionstest 101

6.2 Test der graphischen Oberfläche 103
6.2.1 Testumfang 103
6.2.2 Testdurchführung 103
6.2.3 Auswertung 103
6.2.3.1 Funktionstest 103
6.2.3.2 Usability-Studie 104

7 ZUSAMMENFASSUNG UND AUSBLICK ZU DIESER ARBEIT 105

7.1 Zusammenfassung 105

7.2 Ausblick 106

A ABBILDUNGSVERZEICHNIS 107

B TABELLENVERZEICHNIS 109

C LITERATURVERZEICHNIS 110

D ZUSAMMENFASSUNG DER WICHTIGSTEN LITERATURQUELLEN 118

E SOFTWAREQUELLEN 119

F BEDIENUNGSANLEITUNG FÜR VISUAL L 121

G CD-ROM INHALT 123

H QUELLTEXTE 124

1 Einleitung zu dieser Diplomarbeit

1.1 Motivation zur Modellierung von Pflanzen mit L-Systemen

Die Vielfalt von Pflanzen und deren komplexe (variantenreiche) Strukturen scheinen nur schwer beschreibbar zu sein. Wenn man sich zum Beispiel die Anordnung der Samen, auch Phyllotaxis genannt, einer Sonnenblume betrachtet, ist auf den ersten Blick zu erkennen, dass diese kreisförmig angeordnet sind und es dadurch erschwert wird, eine genau Beschreibung zu erstellen. Wenn man jedoch einer streng mathematischen Beschreibung folgt, ermöglicht es diese, eine genaue Anordnung der Samen wiederzugeben. Es hat sich gezeigt, dass der Verteilung der Samen ein Optimierungsprozess zugrunde liegt, der die runde Fläche am besten ausnutzt. Dabei folgt die mathematische Beschreibung der Anordnung des goldenen Schnittes. [De03]

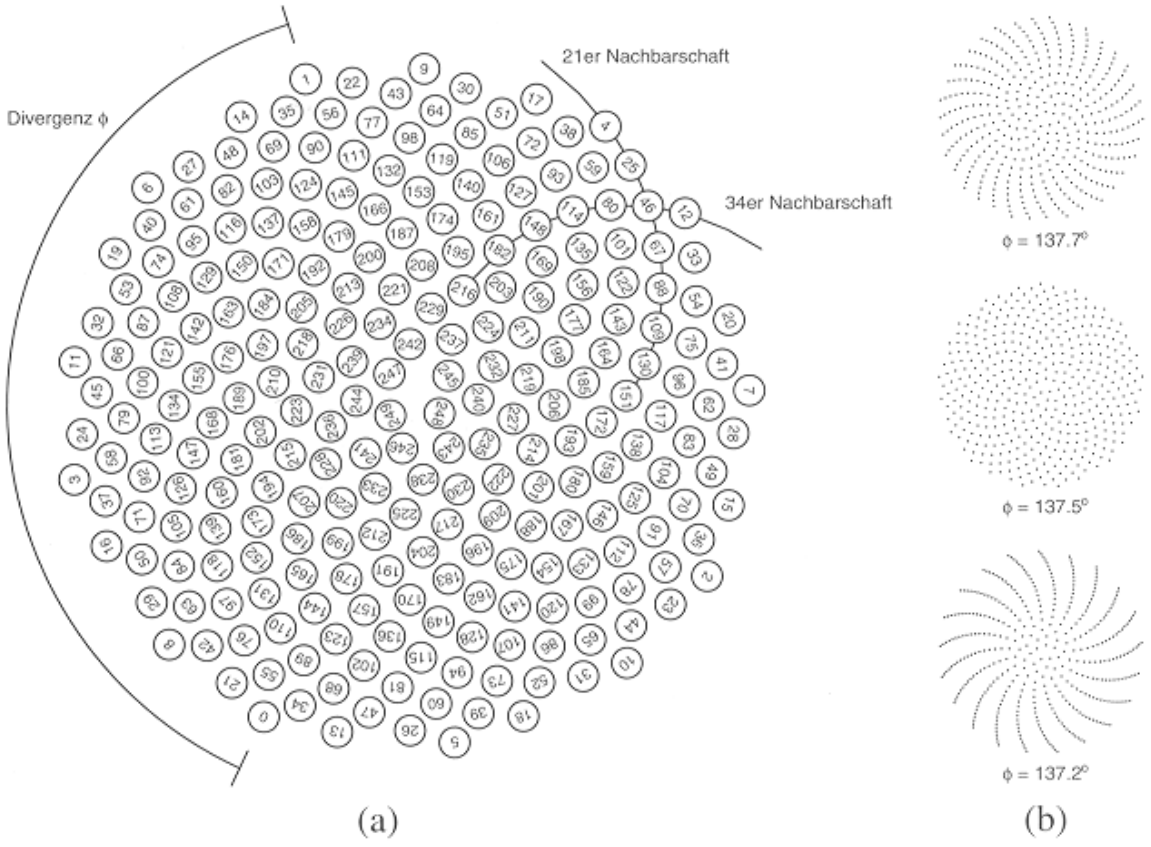


Abbildung 1.1: Phyllotaxis der Sonnenblume [De03]

Ebenso verhält es sich mit der Struktur von Pflanzen. Die Verästelung der Zweige geschieht nicht willkürlich, sondern ebenfalls nach einer bestimmten Gesetzmäßigkeit.
So findet man bei der Knospe drei verschiedene Blattstellungen.
Dabei nennt man den ringförmigen Bereich, an dem die Knospe entsteht, Knoten (Nodus). Zwischen zwei Knoten liegt ein Bereich, der Internodium heißt, an dem keine Knospe entsteht.

Die erste Stellung (vgl. Abbildung 1.2 a) nennt man Distichie. Diese Gesetzmäßigkeit sagt aus, dass pro Knoten nur eine Knospe existiert und dass die Knospen immer um 180° versetzt angeordnet sind.
Die zweite Stellung (vgl. Abbildung 1.2 b) ist die Dispersion. In diesem Fall sind die Knospen schraubenförmig angeordnet und haben einen Divergenzwinkel, die den goldenen Schnitt approximiert.
Die letzte Stellung (vgl. Abbildung 1.2 c) ist die Dekussation. Dabei können mehrere Knospen an einem Knoten entstehen. Die Folgeknospen sind so angeordnet, dass sie in der Lücke zwischen zwei Knoten liegen. [De03]

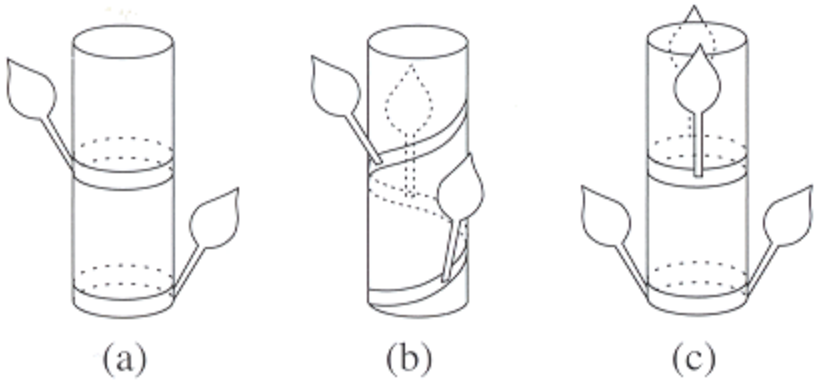


Abbildung 1.2: Stellungen der Knospen [De03]

Dadurch, dass diese Gesetzmäßigkeiten bei Pflanzen existieren, kann man ein mathematisches Modell aufstellen, welches eine bestimmte Pflanze wiedergibt.
In der Computervisualistik hat es sich gezeigt, dass sich die L-Systeme als mathematisches Modell zur Pflanzenmodellierung hervorragend eignen.

Dieses mathematische Modell kann nun mit anderen Elementen der Informatik verknüpft werden. Da Pflanzen einer natürlichen Evolution unterliegen, ist dies eine mögliche Schnittstelle zur künstlichen Intelligenz. So könnte zum Beispiel die Genetische Programmierung genutzt werden, um die natürliche Evolution zu simulieren. Wenn man von einem generierten Wort eines L-Systems ausgeht, könnte das Wachstum rückwärts verfolgt werden, um die Entwicklung einer Pflanze nachzuvollziehen. Das könnte so weit reichen, dass Aussagen darüber gemacht werden können, welchen Bedingungen ein Baum im Laufe eines Jahres ausgesetzt war.
Diese Arbeit ist der Anfang von mehreren Arbeiten, die zusammen ein Konzept ergeben, welches L-Systeme und die künstliche Evolution miteinander verknüpft. Dabei werden Ideen für das Gesamtkonzept in dieser Diplomarbeit behandelt.

1.2 Aufgabenstellung

Ziel der Arbeit ist eine Untersuchung zur automatischen Erstellung von Objektbeschreibungen aus vorgegebenen Quellbildern.

Die Objektbeschreibung erfolgt in Form von L-Systemen, die passend zu Quellbildern erzeugt werden. Hierzu wird ein evolutionärer Prozess auf die L-Systeme angewendet. Fitnessfunktion sei die Übereinstimmung der Ansichten der erzeugten L-Systeme aus verschiedenen Kameraperspektiven mit den Originalbildern eines einfachen Gegenstandes.
Die Arbeit soll die theoretischen Grundlagen dieses Ansatzes darlegen (insbesondere Einordnung und Klassifizierung von L-Systemen und ihre Evolutionsmöglichkeiten), eine Systemarchitektur vorschlagen und in Teilen prototypisch implementieren. Bei der Implementation soll besonderer Wert auf die Weiternutzbarkeit der erstellten Module und die Definition externer Schnittstellen gelegt werden. Die Funktionsfähigkeit des Systems ist durch geeignete Teststellungen zu evaluieren.

1.3 Gliederung

Im folgenden Kapitel werden die theoretischen Grundlagen für diese Arbeit erläutert. Dazu wird das ursprünglich mathematische Modell von Lindenmayer vorgestellt und dessen wichtigste Unterschiede zu den Sprachen der Chomskyhierarchie aufgezeigt. Des Weiteren werden die wichtigsten L-Systeme besprochen, wobei kurz auf einige spezielle L-Systeme eingegangen wird. Danach werden die Klassifikationen besprochen sowie die Interpretation des Wortes in graphische Kommandos. Außerdem wird gezeigt, wie L-Systeme eingesetzt werden. Auch auch alternative Sprachen zu den L-Systemen werden kurz vorgestellt. .
Zusätzlich zu den Grundlagen der L-Systeme wird eine Übersicht zu den Genetischen Algorithmen und zur Genetischen Programmierung gegeben, damit später die Konzepte zur Mutation von L-Systemen diskutiert werden können.
Zum Abschluss des Kapitels werden die Anwendungsmöglichkeiten dieser Arbeit besprochen.

Das dritte Kapitel gibt einen Überblick über vorhandene Applikationen zu L-Systemen, wobei einige kurz erläutert werden. Am Ende werden tabellarisch die wichtigsten Informationen zusammengestellt.

Das grundlegende Konzept und der Entwurf für die Software der Diplomarbeit sind Inhalt des vierten Kapitels. Hier wird die Entwicklungsumgebung und die benutzten Tools vorgestellt. Einige Gesichtspunkte aus der Sicht des Software-Engineerings werden dabei genauer betrachtet. Außerdem wird die Implementierung der Software beschrieben.

Die graphische Oberfläche wird im fünften Kapitel dargestellt. Dabei werden die wichtigsten Formulare und die externen Bibliotheken vorgestellt.

Im sechsten Kapitel wird der Schwerpunkt auf die Tests gelegt. Es wird nicht nur die Applikation selbst auf ihre Tauglichkeit getestet, sondern auch deren Performance und die Belastung für den Computer.

Den Abschluss der Arbeit bildet das siebente Kapitel. In diesem wird eine Zusammenfassung der Arbeit vorgenommen und ein Ausblick auf die mögliche Erweiterung der Software gegeben.

Im Anhang befinden sich das Literaturverzeichnis, das Abbildungsverzeichnis, das Tabellenverzeichnis und ein Verzeichnis über die Softwarequellen. Des Weiteren ist eine Übersicht der Baumstruktur der CD-ROM enthalten.

1.4 Anforderungen an den Leser

Im Rahmen dieser Arbeit werden einige Anforderungen an den Leser gestellt, denn sie reißt viele Gebiete der Informatik an, auf die aus Platzgründen im Einzelnen nicht eingegangen werden kann.
Hierzu zählt ein gewisser Grad an mathematischem Verständnis, der spätestens mit dem Erreichen des Vordiploms erworben sein sollte. Zum Nachschlagen sei auf folgende Quellen verwiesen [SGT99] [SD01] [Gö99].
Der Leser sollte über die Grundlagen der Formalen Sprachen verfügen und die Chomskyhierarchie sowie die Grundlagen zur Automatentheorie kennen. Zum Nachlesen empfiehlt sich folgende Literatur [HU00] [Wä99].
Kenntnisse über Datenstrukturen sollten auch vorhanden sein und können in [Se92] [OW02] nachgeschlagen werden.
Für den praktischen Teil der Arbeit werden allgemeine Kenntnisse der Programmierung vorausgesetzt, insbesondere Kenntnisse der Programmiersprachen C und Delphi. Folgende Literatur kann genutzt werden, um das nötige Wissen zu erhalten [KR90] [PP99] [Wi95] [DK00] [Lo00] [Eb02]. Nicht nur als Übungsbuch für Delphi, sondern auch als Lehrbuch für bessere und fortgeschrittene Programmiertechniken ist [El00] [El01] zu empfehlen.
Eine weitere notwendige Grundlage ist das Wissen über Software-Engineering, das zum Beispiel aus [Ba98] [Ba99] [Ba00] entnommen werden kann.
Grobe Grundlagen der graphischen Datenverarbeitung, im Speziellen von 3D-Grafiken in Verbindung mit Ray-Tracing, sind vorteilhaft und können in [ESK96] [ESK97] [GFMP01] nachgelesen werden.

2 Theoretische Grundlagen von L-Systemen und deren Evolution

Dieses große und umfassende Kapitel vermittelt die theoretischen Grundlagen, die für diese Arbeit nötig sind. Es wird nicht nur auf die verschiedenen L-Systeme eingegangen, sondern auch auf Mutationskonzepte von L-Systemen unter Nutzung von Genetischer Programmierung und Genetischen Algorithmen. Das Kapitel zeigt auch Alternativen zu L-Systemen auf sowie die Anwendungsmöglichkeiten von L-Systemen in Kombination mit der Evolution. Zuvor werden jedoch kurz die drei Personen vorgestellt, die für diesen Bereich prägend sind.

2.1 Kurzbiografien von wichtigen Personen

Aristid Lindenmayer: [FMF04]
Aristid Lindenmayer kommt am 17. November 1925 in Budapest zur Welt. Dort studiert er auch und erhält seinen Abschluss in Chemie. Er wandert in die USA aus und wird wissenschaftlicher Mitarbeiter an der Universität Michigan, wo er in Biochemie promoviert. Danach verlässt er Michigan und ist zeitweise an der Universität von Pennsylvania und am Queens College in New York tätig. Am 23. August 1967 schreibt Lindenmayer den Artikel über sein mathematisches Modell, welches dann nach ihm benannt wird, das Lindenmayer-System [Li68a], [Li68b]. Diesen Artikel veröffentlicht er noch am Queens College. 1968 geht er dann in die Niederlande, um die Position des Direktors für Theoretische Biologie an der Universität Utrecht zu übernehmen. Dieses Amt übt er bis zu seinem Tod im Jahre 989 aus.[1]

Przemyslaw Prusinkiewicz: [Si97]
Sein Geburtsort und das Geburtsdatum sind unbekannt. 1974 schließt Prusinkiewicz sein Studium mit dem Master of Science an der Technischen Universität Warschau ab. Hier bleibt er als wissenschaftlicher Mitarbeiter bis zu seiner Promotion im Jahre 1979. Danach geht er für drei Jahre nach Algerien, wo er an der Universität von Algier arbeitet. 1982 wandert er in die USA aus und wird wissenschaftlicher Mitarbeiter an der Universität Regina. Während dieser Zeit arbeitet Prusinkiewicz an der Visualisierung von Strukturen und demWachstum von Pflanzen. 1986 präsentiert er zum ersten Mal seine Ergebnisse auf der Konferenz „Graphics Interface“ [Pr86]. Seit 1991 lebt er in Kanada. Dort ist er Professor an der Universität Calgary im Fachbereich Informatik. . Außerdem ist er Gastprofessor und Gastforscher an vier Universitäten.

Grzegorz Rozenberg: [Ro04]
Auch der Geburtsort und das Geburtsdatum von Rozenberg sind unbekannt. Wie Prusinkiewicz studiert Rozenberg an der Technische Universität Warschau. Er schließt sein Studium 1965 mit dem „Master and Engineer degree in computer science“ ab. Drei Jahre später erhält er den Doktortitel an der „Polish Academy of Sciences“ in Warschau. Bis 1979 ist er an vier Universitäten Professor, seit diesem Zeitpunkt ist er Professor an der Universität Leiden im Fachbereich Informatik und lebt auch dort. Er erhält 1997 den Gödel Preis und ist an zwei Universitäten Honorar-Doktor, unter anderem an der TU Berlin. Rozenberg entwickelte zu den L-Systemen viele theoretische Grundlagen und brachte dadurch die L-Systeme entschieden weiter.

[1] Es sei darauf hingewiesen, dass viele Angaben zur Person im Internet falsch sind. Sei es z. B. sein Geburtsland, Sterbe- oder Geburtsdatum.

2.2 L-Systeme

Es werden nun die wichtigsten Aspekte der L-Systeme dargestellt. Dazu gehören die wichtigsten L-Systeme selbst sowie die Interpretation des Wortes eines L-Systemes in graphische Kommandos. Außerdem wird ein Blick auf die Klassifikation von L-Systemen in Bezug auf die Chomskyhierarchie gegeben sowie ein Entscheidungsautomat für die Computergrafik präsentiert. Das Kapitel endet mit einem kurzen Überblick über die Anwendungsmöglichkeiten von L-Systemen.

2.2.1 Wichtige Unterschiede zu den Chomskysprachen

Um einen Einstieg in die L-Systeme zu bekommen wird gezeigt, welche Unterschiede zwischen den L-Systemen und den Sprachen der Chomskyhierarchie existieren. Deshalb wird hier auf Kapitel 2.2.3 vorgegriffen und ein Beispiel für ein L-System präsentiert. Die genaueren Definitionen folgen dann dort.

Im Beispiel 2.1 wird eine Grammatik G für ein L-System definiert. Dabei steht Σ für die Menge aller Symbole, die in dieser Grammatik verwendet werden, P für die Menge aller Produktionen und ω für das Axiom, mit dem gestartet wird.

Beispiel 2.1 für ein einfaches L-System:
Das Beispiel zeigt ein einfaches, triviales L-System.

$$G = (\quad \Sigma = \{a\},\quad P = \{a \to aa\},\quad \omega = a \quad)$$

Die Ableitung der ersten fünf Schritte, entwickelt folgendes Wort:

$a \Rightarrow aa \Rightarrow aaaa \Rightarrow aaaaaaaa \Rightarrow aaaaaaaaaaaaaaaa$ oder auch kurz $a \overset{4}{\Rightarrow} a^{16}$.

L-Systeme haben zwei wichtige Eigenschaften, welche die wesentlichen Unterschiede zu den Chomskysprachen beinhalten.
Einer dieser Unterschiede ist, dass L-Systeme nicht zwischen Terminalzeichen und Nichtterminalzeichen unterscheiden (mit Ausnahme der E0L-Systeme). Das Wichtige an den Zeichen ist, dass diese nur enthalten sein dürfen, wenn für sie eine Ableitung existiert.
Der zweite Unterschied liegt bei den Ableitungen. Bei Sprachen der Chomskyhierarchie ist die Ableitung sequenziell. Jeder Ableitungsschritt besteht nur aus der Ableitung einer Produktion. Die L-Systeme führen Ableitungen parallel aus. So wird bei jedem Ableitungsschritt für jedes Zeichen eine Produktion ausgeführt. Damit einher geht auch das Problem des Wachstums. Das Wachstum eines zu generierenden Wortes nimmt bei L-Systemen, die komplexe Objekte beschreiben sollen, in der Regel mindestens exponentiell zu.
An Beispiel 2.1 wird deutlich, dass die Berechnungskomplexität für ein triviales Beispiel schon bei $O(n) = 2^n$ liegt. Bei der Modellierung von Pflanzen kann die Berechnungskomplexität noch um vieles höher sein. Daher muss die Anzahl der Ableitungsschritte bei der synthetischen Modellierung von Pflanzen gut durchdacht sein, damit der Computer in einer endlichen Zeit terminiert. Das Problem der Komplexität von L-Systemen ist nicht immer so einfach zu lösen. Alternativ sei auf folgende Quelle [Ke86] verwiesen, in der das Thema ausführlicher behandelt wird. Sie enthält auch weitere Quellen zum Thema Komplexität und L-Systeme.

L-Systeme in eine bestimmte Sprachhierarchie einzugliedern erweist sich als sehr schwer. Es existieren einige Dissertationen [Gä96], die einzelne L-System in die Chomskyhierarchie einzuordnen versuchen. Diese Einordnung kann jedoch nur für wenige L-Systeme (E0L-Systeme) problemlos erfolgen, bei anderen (S0L-Systeme) ist dies nur sehr schwer möglich. Hopcroft und Ullman [HU00] ordnen die L-Systeme den indizierten Sprachen zu.

2.2.2 Das ursprüngliche L-System

Nach vielen Jahren der Forschung über Bäckerhefe, siehe Referenzliste der Publikationen von Lindenmayer in [RS86], stellte Aristid Lindenmayer 1966 auf dem vierten Symposium für Biomathematik und Informatik in Biowissenschaften in Houston sein mathematisches Modell vor. 1968 wird sein Artikel zum mathematischen Modell gedruckt, auf das nun eingegangen werden soll. Von Interesse ist in diesem Zusammenhang, dass Lindenmayer erst 1971 in *„Developmental Systems without cellular interactions, their languages and grammars“* die Grammatiken für sein Entwicklungssystem eingeführt [RS86]. Das ursprüngliche Modell basiert auf Automaten und noch nicht auf Grammatiken.
Als Lindenmayer an seinem mathematischen Modell arbeitet, sind die aktuellen Themen in der Entwicklungsbiologie die Kontrolle der Zellteilung, die Differenzierung von Zellen und die Zellausdehnung. Getrieben von dem Gedanken, dass diese Operationen vom gesamten Organismus selbst kontrolliert werden können, entwickelt er sein mathematisches Framework. Lindenmayer erkennt dass es nicht mehr möglich ist, alle Kombinationen zu beherrschen, wenn man über die Entwicklung von einer Hand voll Zellen hinausgehen möchte.
Das erste Beispiel zeigt einen endlichen Automaten, kurz EA, der ein lineares Array von Zellen verarbeiten soll.

Beispiel 2.2 für EA mit einseitiger Eingabe ohne Wachstum [Li68a]:
Das lineare Array von Zellen sieht im Anfangszustand wie folgt aus.

0 1 1 0 0

Der EA hat nur zwei Zustände, 0 und 1, wobei die Ausgabe gleich dem Zustand ist. Der Anfangszustand des EA ist abhängig von dem Zustand der ersten Zelle auf der linken Seite. Endzustände sind sowohl 0 als auch 1. Das Eingabealphabet besteht ebenfalls aus 0 und 1. δ ist in der Tabelle 2.1 und im Transitionsdiagramm in Abbildung 2.1 beschrieben.

Zustände	Eingabe	
	0	1
0	0	1
1	1	0

Tabelle 2.1: Übergangsmatrix für den EA

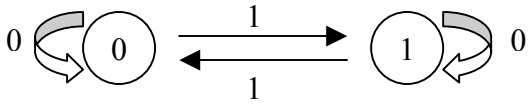


Abbildung 2.1: Transitionsdiagramm für den EA

Um den Automaten zu betreiben, wird noch eine willkürliche linke Eingabe benötigt. Lindenmayer nennt diese Eingabe „environmental input“. In diesem Beispiel soll es 1 sein. Es ergibt sich folgende Zustandsänderung des Zellenarrays.

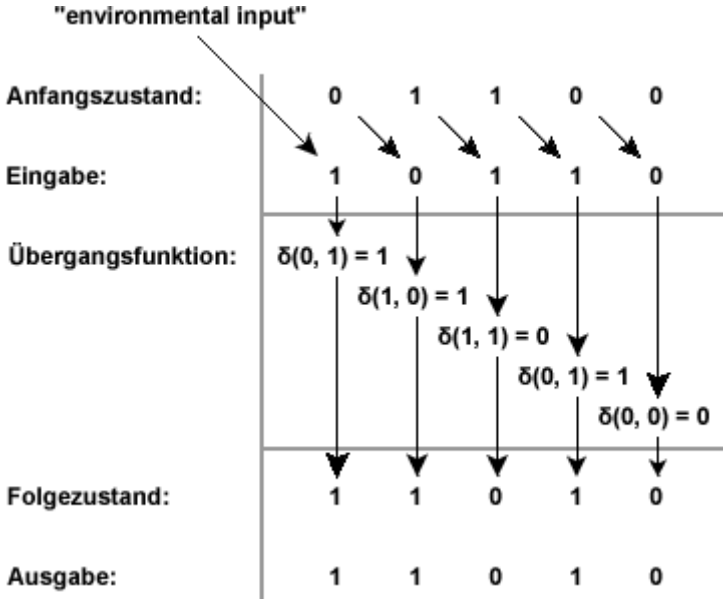


Abbildung 2.2: Übergang vom Anfangszustand zum ersten Folgezustand

Das Beispiel zeigt ein einfaches Verhalten für Zellenarrays. Das Problem ist, dass diese nur zwischen zwei Zuständen wechseln können. Die Zellen können sich nicht teilen und auch nicht absterben. Das Zellenarray kann zwar nicht sterben, aber sich auch nicht weiterentwickeln. Das nächste Beispiel zeigt, wie durch eine kleine Veränderung die Möglichkeit der Zellteilung hinzugefügt wird.

Beispiel 2.3 für EA mit einseitiger Eingabe und mit Wachstum [Li68a]:
Alle Eigenschaften des EA aus dem Beispiel 2.1 werden übernommen, bis auf die Übergangsfunktion. Diese wird in Tabelle 2.2 beschrieben.

Zustände	Eingabe	
	0	1
0	0	1
1	1 1	0

Tabelle 2.2: Übergangsmatrix für den EA

In der folgenden Abbildung wird der Übergang vom Anfangszustand in den Folgezustand illustriert.

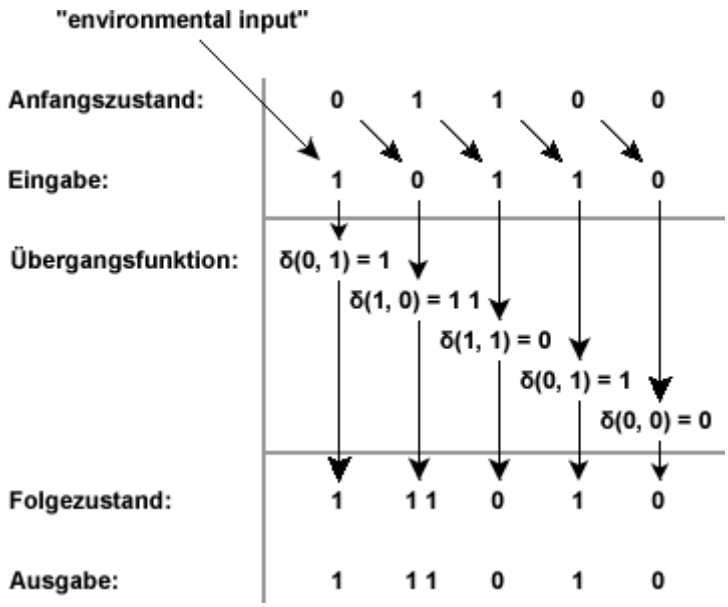


Abbildung 2.3: Einführung der Zellteilung und der entsprechende Folgezustand

Die EA aus den Beispielen ähneln dem Verfahren für die zellulären Automaten (weitere Informationen dazu in [Ja97] [De03]), haben aber gravierende Unterschiede. So hat das Array keine feste Größe, sondern kann wachsen aber auch schrumpfen. Des Weiteren ist die Zustandsänderung bei einseitiger Eingabe nur abhängig von dem linken Kontext und nicht, wie bei zellulären Automaten, abhängig vom linken und rechten Kontext. Außerdem ist der „environmental input“ ausschlaggebend für die Entwicklung der Zellen. Solch eine Eingabe existiert bei den zellulären Automaten ebenfalls nicht.
Diese beiden Beispiele zeigen die einfachste Implementierung des mathematischen Modells. Ausgelegt sind diese Beispiele für einseitige Eingaben. Aus biologischer Sicht haben eindimensionale, fadenförmige Organismen zwei Seiten, von denen aus ein Organismus Eingaben wahrnehmen kann. Dazu führt Lindenmayer die Vektorpfeile über der Übergangs- und Ausgabefunktion ein. So gilt $\vec{\delta}$ für eine linksseitige Übergangsfunktion und $\overleftarrow{\delta}$ für eine rechtsseitige Übergangsfunktion. Analog gilt es auch für die Ausgabefunktion λ. Lindenmayer führt auch die Verzweigung in sein Entwicklungssystem ein. Um den Anfang einer Verzweigung zu beschreiben, wird das Symbol „[“ verwendet. Das Ende einer Verzweigung wird entsprechend mit „]“ beschrieben. [Li68b]

2.2.3 Übersicht der L-Systeme

Nachdem das ursprüngliche mathematische Modell vorgestellt ist, werden jetzt die geläufigen L-Systeme erläutert. Die L-Systeme sind als Grammatik definiert und nicht, wie ursprünglich, als Automat. Es werden die wichtigsten L-Systeme besprochen und Beispiele für deren Grammatik und Anwendung gegeben. Einige spezielle L-Systeme, die nicht unbedingt interessant für die Computergrafik sind, werden nur angerissen.

2.2.3.1 Basis L-System

Das 0L-System ist die Basis aller kontextfreien L-Systeme. Dabei steht die Null, nicht O (der Buchstabe), wie in vielen Quellen falsch angegeben, vor dem L für ein kontextfreies L-System, bzw. für ein L-System ohne Interaktion. Es folgen die wichtigsten Definitionen für 0L-Systeme, die auch für alle anderen kontextfreien L-Systeme gelten, es sei denn, es wird explizit darauf hingewiesen, dass andere Definitionen für ein anderes L-System gelten.

Definition 2.1 für ein 0L-Schema [HR75]:
Ein 0L-Schema ist ein Paar $S = \langle \Sigma, P \rangle$, wo Σ (das Alphabet von S) eine endliche, nicht leere Menge ist und P (die Menge der Produktionen von S) eine endliche, nicht leere Teilmenge von $\Sigma \times \Sigma^*$ ist, so dass

$$(\forall a)_{\Sigma} (\exists \alpha)_{\Sigma^*} (\langle a, \alpha \rangle \in P). \tag{2.1}$$

Jedes Symbol aus Σ muss mindestens eine Produktion in der Menge der Produktionen P aufweisen. Dabei ist es ausgeschlossen, dass ε, das leere Wort, auf der linken Seite einer Produktion steht.

Definition 2.2 für die direkte Ableitung [HR75]:
Sei $S = \langle \Sigma, P \rangle$ ein 0L-Schema, sei $x = a_1 ... a_m$ mit $m > 0$ und $a_j \in \Sigma$ für $j = 1,...,m$ und sei $y \in \Sigma^*$. Dann sagt man, dass *x direkt y (in S) ableitet*, und deutet es mit $x \underset{S}{\Rightarrow} y$ an, aber auch nur genau dann, wenn

$$(\exists \alpha_1,...,\alpha_m)_{\Sigma^*} \left(\left(a_1 \underset{P}{\rightarrow} \alpha_1,..., a_m \underset{P}{\rightarrow} \alpha_m \right) und \left(y = \alpha_1,..., \alpha_m \right) \right). \tag{2.2}$$

Definition 2.3 für die endliche Sprache [HR75]:
Für ein 0L-Schema $S = \langle \Sigma, P \rangle$, für ein Wort x in Σ^* und für eine nicht negative ganze Zahl n wird die endliche Sprache $L_n(S, x)$ durch Induktion über n definiert.

$$L_0(S, x) = \{x\}, \tag{2.3}$$

$$L_{n+1}(S, x) = \left\{ y \,\middle|\, (\exists z) \left(z \in L_n(S, x) \; and \; z \underset{S}{\Rightarrow} y \right) \right\}. \tag{2.4}$$

Definition 2.4 für ein 0L-System [HR75]:
Ein 0L-System ist ein Tripel $G = \langle \Sigma, P, \omega \rangle$, wo $S = \langle \Sigma, P \rangle$ das 0L-Schema ist (auch das Schema von G genannt) und ω (das Axiom von G) ist ein Wort über Σ.

Definition 2.5 für L(G) [HR75]:
Sei $G = \langle \Sigma, P, \omega \rangle$ ein 0L-System. Die Sprache, die *G generiert* (oder einfach die Sprache von G), bezeichnet mit $L(G)$, ist definiert als

$$L(G) = \left\{ x \,\middle|\, w \overset{*}{\underset{G}{\Rightarrow}} x \right\}. \tag{2.5}$$

Es folgt ein Beispiel zum besseren Verständnis von 0L-Systemen. Dabei steht das Symbol | für eine alternative Auswahl und ist nicht Bestandteil des Alphabetes Σ.

Beispiel 2.4 für ein 0L-System:
Für ein 0L-System sei folgende Grammatik definiert.

$$
\begin{aligned}
G = (\quad & \Sigma = \{\quad a,b,c,d,S,\varepsilon \quad\}, \\
& P = \{\quad S \to ab, \\
& \qquad a \to a \mid bc, \\
& \qquad b \to bda \mid d, \\
& \qquad c \to \varepsilon, \\
& \qquad d \to aba \qquad\}, \\
& \omega = \quad S \qquad\qquad\qquad)
\end{aligned}
$$

Für die ersten drei Ableitungsschritte ergibt sich folgender Ableitungsbaum.

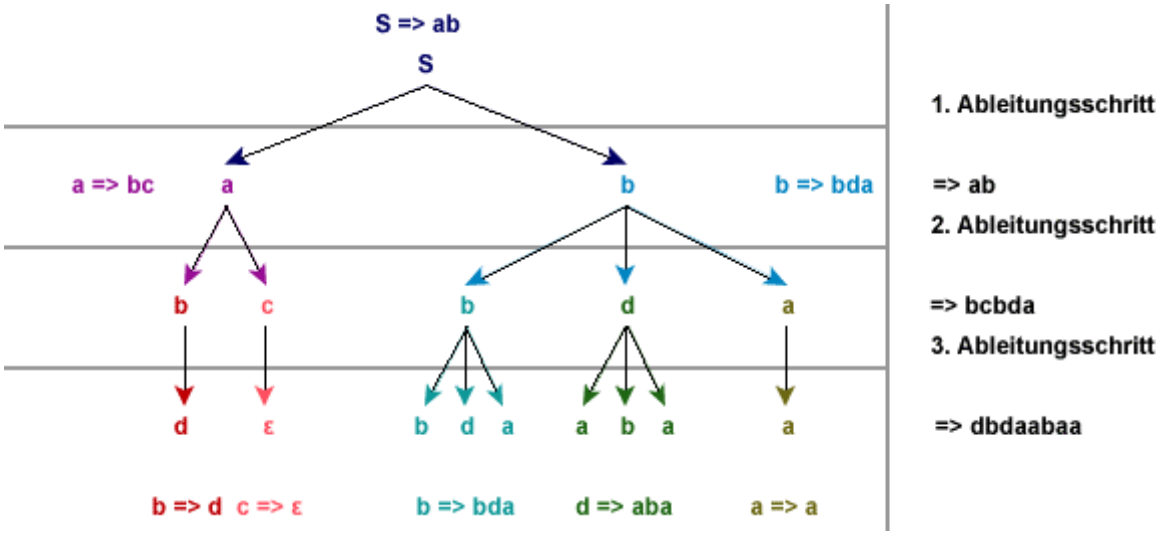


Abbildung 2.4: Ableitungsbaum zum Beispiel 2.4

Dementsprechend ist das Wort, welches aus diesem 0L-System generiert wird,
$d\varepsilon\ b\,d\,a\,a\,b\,a\,a = d\,b\,d\,a\,a\,b\,a\,a$.

Anwendungsbereiche:
Diese einfache Familie der L-Systeme eignet sich für jegliche Art von fadenförmigen Organismen [Li68a] [Li68b]. Das hängt damit zusammen, dass Lindenmayer dieses mathematische Modell für solche Organismen entwickelt hat. So kann jedes Zeichen aus dem Alphabet der Grammatik eine Zelle in einem bestimmten Zustand abbilden, oder jedes Zeichen eine andere Art von Zelle darstellen. Mit jedem Ableitungsschritt erkennt man, wie sich der Organismus weiterentwickelt. Dabei kann sich eine Zelle teilen, die Zelle bleibt erhalten oder eine Zelle stirbt ab. Das Absterben kann mit dem leeren Wort simuliert werden. Es ist klar, dass bei fadenförmigen Organismen die Betrachtung in der ersten Dimension geschieht. Daher kann das Wort direkt interpretiert werden und ist auch leicht verständlich. Mit der späteren Einführung einer Interpretation des Wortes (vgl. Kapitel 2.2.4) können andere Objekte betrachtet werden, wie zum Beispiel Pflanzen, Gegenstände und Kreaturen [Pr93] [Pr86] [PL90] [PJM94] [PHHM96b] [PHHM96a] [PHHM95] [BPFGK03] [HLP01] [Ho03] [HP01a] [HP01b] [HP01c] [HP02] [GR92].

2.2.3.2 Deterministisches L-System

Das D in solch einem D0L-System steht für *deterministic* und bedeutet, dass für jede Ableitung eines Zeichen nur eine Möglichkeit der Ableitung existiert.

Definition 2.6 für ein D0L-Schema [HR75]:
Ein 0L-Schema $S = \langle \Sigma, P \rangle$ ist *deterministisch*, wenn für alle a in Σ exakt ein α in Σ^* der Form $a \underset{P}{\rightarrow} \alpha$ existiert. Andernfalls, nennt sich *S nicht deterministisch*.

Definition 2.7 für ein D0L-System [HR75]:
Ein 0L-System $G = \langle \Sigma, P, \omega \rangle$ ist deterministisch genau dann, wenn *S* deterministisch ist.

Das 0L-System aus dem Beispiel 2.4 ist nicht deterministisch, weil für die Zeichen *a* und *b* mehrere Möglichkeiten der Ableitung existieren. So kann schon nach dem zweiten Ableitungsschritt ein Wort wie *ad* entstehen, weil statt der Produktion $a \rightarrow bc$ die Produktion $a \rightarrow a$ gewählt wurde und statt der Produktion $b \rightarrow bda$ die Produktion $b \rightarrow d$.

Beispiel 2.5 für ein D0L-System [HR75]:
Für G sei folgendes D0L-System definiert.

$$
\begin{aligned}
G = (\quad & \Sigma = \{\quad a, b, c, d, S, \varepsilon \quad\}, \\
& P = \{\quad S \rightarrow ab, \\
& \qquad\quad a \rightarrow bc, \\
& \qquad\quad b \rightarrow bda, \\
& \qquad\quad c \rightarrow \varepsilon, \\
& \qquad\quad d \rightarrow aba \qquad\}, \\
& \omega = \quad S \qquad\qquad\qquad)
\end{aligned}
$$

Auswirkungen:
Ist ein 0L-System deterministisch, so bedeutet dies, dass eine Zelle nicht mehrere Aktionen ausführen kann. Um zum Beispiel abzusterben, muss sich die Zelle mittels einer Produktion in eine andere Art von Zelle transformieren, die als einziges Zeichen auf der linken Seite das ε hat.

2.2.3.3 Propagierendes L-System

Das P in einem P0L-System steht für *propagating*. Damit ist gemeint, dass in keiner Produktion des L-Systems das ε auftauchen darf.

Definition 2.8 für ein P0L-Schema [HR75]:
Ein 0L-Schema $S = \langle \Sigma, P \rangle$ heißt propagierend, wenn es keine Produktion in *P* der Form $a \rightarrow \varepsilon$ gibt. (Eine Produktion dieser Form nennt man *löschende Produktion.*)
Andernfalls nennt man *S nicht propagierend*.

Definition 2.9 für ein P0L-System [HR75]:
Ein 0L-System $G = \langle \Sigma, P, \omega \rangle$ ist propagierend genau dann, wenn *S* propagierend und $\omega \neq \varepsilon$ ist.

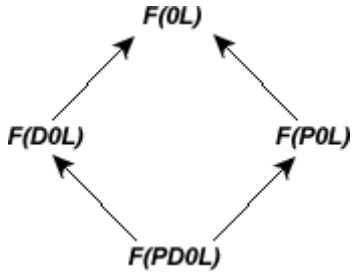
Die in Beispiel 2.4 definierte Grammatik *G* für ein 0L-System ist nicht propagierend, weil das Zeichen *c* eine löschende Produktion darstellt, welches gegen die Definition 2.8 verstößt. Folgendes Beispiel stellt ein P0L-System dar.

Beispiel 2.6 für ein P0L-System:

$$
\begin{aligned}
G = (\quad & \Sigma = \{\quad a, b, c, d, S, \varepsilon \quad\}, \\
& P = \{\quad S \to ab, \\
& \qquad a \to a \mid bc, \\
& \qquad b \to bda \mid d, \\
& \qquad c \to c, \\
& \qquad d \to aba \qquad\}, \\
& \omega = \quad S \qquad\qquad\qquad)
\end{aligned}
$$

Satz 2.1 über den Zusammenhang der L-Systeme:
Der gerichtete Graph soll folgenden Sachverhalt zeigen:

$$F(PD0L) \subset F(D0L), F(D0L) \subset F(0L),$$
$$F(PD0L) \subset F(P0L) \text{ und } F(P0L) \subset F(0L)$$



Zwei Sprachfamilien, die nicht über einen Pfad erreichbar sind, sind zueinander inkompatibel aber nicht disjunkt.

Beweis:
[HR75]

Auswirkungen:
Ein P0L-System ist ein lebendes System, das heißt kein Zeichen kann durch die Anwendung einer Produktion aus dem Wort gelöscht werden. Für einen fadenförmigen Organismus heißt es, dass Zellen sich nur teilen können oder bestehen bleiben. Keine Zelle kann in einem lebenden System absterben. Der Organismus entwickelt sich immer weiter.

2.2.3.4 L-Systeme mit Terminalzeichen

Ein E vor einem E0L-System, steht für *Extension*.
Die *Extension* in solch einem 0L-Systeme ist die Einführung von Terminalzeichen. Durch die E0L-Systeme erhält man eine Schnittstelle zu den Sprachen der Chomskyhierarchie. Im Gegensatz zu den Sprachen der Chomskyhierarchie können Terminalzeichen weiter abgeleitet werden. Des Weiteren ist ein Wort in einer Sprache nur enthalten, wenn in einem Ableitungsschritt alle Zeichen des Wortes Terminalzeichen sind.

Definition 2.10 für ein E0L-System [HR75]:
Ein *E0L-System* ist ein Quadrupel der Form $G = \langle \Sigma, P, \omega, \Delta \rangle$, wo $\overline{G} = \langle \Sigma, P, \omega \rangle$ ein 0L-System ist und $\Delta \subset \Sigma$. Die *Sprache* von *G* (bezeichnet mit $L = L(G)$) ist definiert durch $L(G) = L(\overline{G}) \cap \Delta^*$. In solch einem Fall bezeichnet *L(G)* eine *E0L-Sprache*.

Definition 2.11 für ein volles E0L-System [HR75]:
Ein E0L-System $G = \langle \Sigma, P, \omega, \Delta \rangle$ und seine Sprache *L(G)* wird *voll* genannt genau dann, wenn $\Delta = \Sigma$.

Beispiel 2.7 für ein E0L-System angelehnt an [HR75]:
Die Grammatik eines PE0L-System sei wie folgt definiert:

$$
\begin{aligned}
G = (\quad & \Sigma = \{\quad S, A, B, A', B', F, a, b \quad\}, \\
& P = \{\quad S \to AB, \\
& \qquad A \to AA' \mid a, \\
& \qquad B \to BB' \mid b, \\
& \qquad A' \to A' \mid a, \\
& \qquad B' \to B' \mid b, \\
& \qquad F \to F, \\
& \qquad a \to F, \\
& \qquad b \to F \qquad\quad \}, \\
& \omega = \quad S, \\
& \Delta = \{\quad a, b \qquad\quad \}\quad)
\end{aligned}
$$

Es generiert die Wörter der Sprache $L = \{a^n b^n \mid n > 0\}$.
Es werden zwei Ableitungsbäume für die ersten 3 Ableitungen gezeigt.

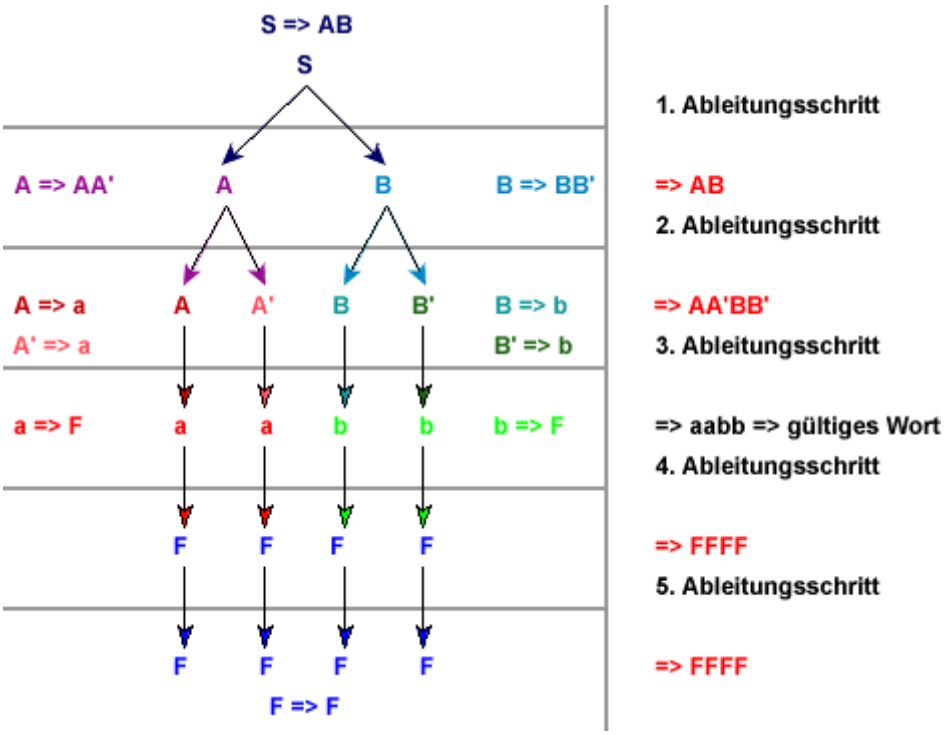


Abbildung 2.5-a: Ableitungsbaum für Beispiel 2.7

Im dritten Ableitungsschritt hat das E0L-System das gültige Wort *aabb* generiert. Es folgt eine alternative Ableitung.

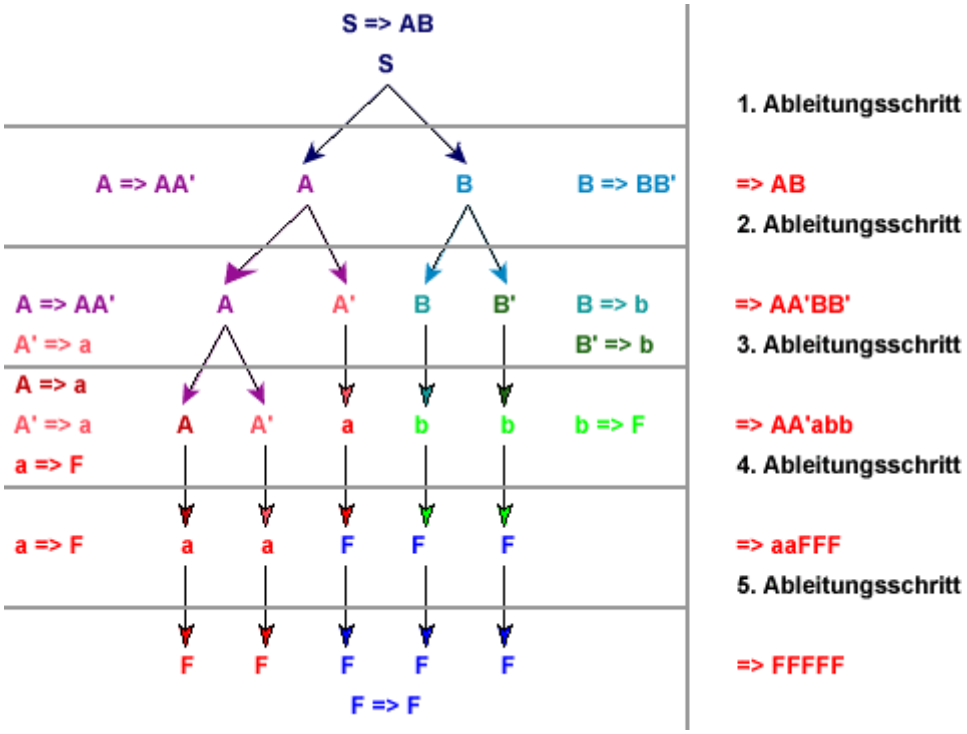


Abbildung 2.5-b: Ableitungsbaum für Beispiel 2.7

Zu keinem Zeitpunkt ist nach einer Ableitung ein Wort erstellt worden, das nur aus Terminalzeichen besteht, daher wurde kein gültiges Wort für diese Sprache entwickelt.

2.2.3.5 L-Systeme mit Tabellen

Das T in einem T0L-System steht für *table*.
Bis jetzt waren alle 0L-Systeme in einer idealisierten Umgebung. Die Zellen eines Organismus haben sich geteilt, sind abgestorben oder einfach erhalten geblieben. Dabei spielte die Umgebung, in der sie leben, keine Rolle. In der realen Welt verhält es sich anders. Hier entwickelt sich jeder Organismus so, wie seine Umgebung es ihm vorgibt. So wächst eine Pflanze beispielsweise nicht, wenn sie kein Wasser oder keine Sonnenstrahlen erhält. ,
Um verschiedene Umgebungssituationen in L-Systemen zu berücksichtigen, wurden T0L-Systeme eingeführt. Jede Tabelle eines T0L-Systems steht für eine bestimmte Umgebungssituation, zum Beispiel für eine Umgebung mit Produktion, wenn es warm ist, kalt, hell oder dunkel. Dabei kann zu jedem Ableitungsschritt nur die Produktionen einer Tabelle benutzt werden.
Durch das Einführen von T0L-Systemen müssen einige Definition für die L-Systeme angepasst werden.

Definition 2.12 für ein T0L-Schema [HR75]:
Ein T0L-Schema ist ein Paar $S = \langle \Sigma, \mathscr{P} \rangle$, wo Σ (das *Alphabet* von *S*) eine endliche, nicht leere Menge und $\mathscr{P}$(die Menge an tables von S) eine endliche, nicht leere Menge ist. Jedes Element *P* von $\mathscr{P}$(auch *table* genannt) ist eine endliche, nicht leere Teilmenge von $\Sigma \times \Sigma^*$ sodass

$$(\forall a)_{\Sigma} (\exists \alpha)_{\Sigma^*} (\langle a, \alpha \rangle \in P). \tag{2.6}$$

Definition 2.13 für die direkte Ableitung [HR75]:
$S = \langle \Sigma, \mathscr{P} \rangle$ sei ein T0L-Schema, $x = a_1,...,a_m$ sei mit $m > 0$ und $a_j \in \Sigma$ für *j = 1, ..., m* und $y \in \Sigma^*$. Dann sagt man, dass *x direkt y (in S) ableitet*, und deutet es mit $x \underset{S}{\Rightarrow} y$ an, aber auch nur genau dann, wenn

$$(\exists P)_{\mathscr{P}} (\exists \alpha_1,...,\alpha_m)_{\Sigma^*} (\{a_1 \underset{P}{\rightarrow} \alpha_1,...,a_m \underset{P}{\rightarrow} \alpha_m\} \, und \, (y = \alpha_1,...,\alpha_m)). \tag{2.7}$$

Definition 2.14 für die endliche Sprache [HR75]:
Für ein T0L-Schema $S = \langle \Sigma, \mathscr{P} \rangle$, für ein Wort *x* in Σ^* und für eine nicht negative ganze Zahl *n* wird die endliche Sprache $L_n(S, x)$ durch Induktion über *n* definiert.

$$L_0(S, x) = \{x\}, \tag{2.8}$$

$$L_{n+1}(S, x) = \{y \mid es\ existiert\ ein\ z\ in\ L_N(S, x) und\ ein\ P\ in\ P\ so\ das\ z \underset{P}{\Rightarrow} y\}. \tag{2.9}$$

Definition 2.15 für ein PT0L-Schema [HR75]:
Ein T0L-Schema $S = \langle \Sigma, \mathscr{P} \rangle$ heißt propagierend, wenn es keine Produktion in jedem *P* von $\mathscr{P}$der Form $a \rightarrow \varepsilon$ gibt (eine Produktion dieser Form nennt man *löschende Produktion*). Andernfalls nennt man *S nicht propagierend*.

Definition 2.16 für ein DT0L-Schema [HR75]:
Ein T0L-Schema $S = \langle \Sigma, \mathscr{P} \rangle$ ist *deterministisch*, wenn für alle *P* in $\mathscr{P}$und für alle *a* in Σ exakt ein α in Σ^* der Form $a \underset{P}{\rightarrow} \alpha$ existiert. Andernfalls nennt sich *S nicht deterministisch*.

Definition 2.17 für ein T0L-System [HR75]:
Ein T0L-System ist ein Tripel $G = \langle \Sigma, \mathscr{P}, \omega \rangle$, wo $S = \langle \Sigma, \mathscr{P} \rangle$ das T0L-Schema ist (auch das *Schema* von *G* genannt) und ω (das *Axiom* von *G*) ein Wort über Σ ist. *G* ist *propagierend* genau dann, wenn *S* propagierend ist und $\omega \neq \varepsilon$. *G* ist *deterministisch* genau dann, wenn *S* deterministisch ist.

Definition 2.18 für L(G) [HR75]:
Sei $G = \langle \Sigma, \mathscr{P}, \omega \rangle$ ein T0L-System. Die Sprache, die *G generiert* (oder einfach die Sprache von G), bezeichnet mit $L(G)$, ist definiert als

$$L(G) = \left\{ x \,\middle|\, w \overset{*}{\underset{G}{\Rightarrow}} x \right\}. \qquad (2.10).$$

Wie schon zu Anfang erwähnt, besteht der Unterschied zwischen einem T0L-Schema und einem 0L-Schema darin, dass mehrere *tables* zur Verfügung stehen und benutzt werden können. Es wird nun definiert, wie dies geschieht.

Definition 2.19 für die Ableitungsfunktion eines T0L-Schemas [HR75]:
Sei $S = \langle \Sigma, \mathscr{P} \rangle$ ein T0L-Schema. Eine Ableitung in *S* ist ein Tripel $D = \langle O, v, p \rangle$, wo *O* eine Menge von geordneten Paaren nicht negativer ganzer Zahlen ist (das *Vorkommen* in *D*), *v* ist eine Funktion von *O* in *Σ* (*v(i, j)* ist der Wert von *D* am Vorkommen $\langle i, j \rangle$) und *p* ist eine Funktion von der Teilmenge *O* in $\mathscr{P} \times \left(\bigcup_{P \in \mathscr{P}} P \right)$ (p(i, j) ist der table-Produktions Wert von D am Vorkommen $\langle i, j \rangle$), die folgenden Bedingungen befriedigen. Es existiert eine Sequenz von Worten $(x_0, x_1, ..., x_n)$ in Σ^* (auch genannt der *trace* von *D* und bezeichnet mit tr(*D*)) so dass

1. $O = \left\{ \langle i, j \rangle \,\middle|\, 0 \le i \le f, 1 \le j \le |x_i| \right\}$,
2. *v(i, j)* ist das *j*te Zeichen in x_i,
3. die Domäne von p ist $\left\{ \langle i, j \rangle \,\middle|\, 0 \le i \le f, 1 \le j \le |x_i| \right\}$,
4. für $0 \le i \le f$, existiert ein P in $\mathscr{P}$, sodass für $1 \le j \le |x_i|$,
   $p(i, j) = \langle P, v(i, j) \to \alpha_j \rangle$, wo $v(i, j) \underset{P}{\to} \alpha_j$ und $\alpha_1 \alpha_2 ... \alpha_{|x_i|} = x_{i+1}$.

In solch einem Fall wird gesagt, dass *D eine Ableitung über* x_f *von* x_0 und *f* die *Höhe* von der Ableitung *D* ist.

Die wichtigsten Eigenschaften sollen nun direkter formulieren werden. *O* bildet Zahlenpaare, wobei die erste Zahl die Ableitung kennzeichnet und die zweite Zahl die Stelle des Wortes (zum Beispiel im Wort *aba* ist an der zweiten Stelle *b*). Während *O* nur abstrakte Zahlenpaare darstellt, kann *O* in *v* eingesetzt werden, um das Zeichen einer Stelle im Wort zu einer bestimmten Ableitung zu bestimmen (beim Beispiel *aba* wäre *v(0, 2) = b*). In p wird auch ein Zahlenpaar von *O* eingesetzt, um die Ableitung für ein bestimmtes Symbol zu erhalten.

Beispiel 2.8 für ein T0L-System [HR75]:
Dieses Beispiel zeigt ein T0L-Schema und die ersten zwei Ableitungsschritte.
Sei $S\langle\{a,b\},\{P_1,P_2\}\rangle$, wobei $P_1=\{a\to a^2, b\to b^2\}, P_2=\{a\to a^3, b\to b^3\}$. Sei
$D=\langle O,v,p\rangle$, wobei $O=\{\langle 0,1\rangle,\langle 0,2\rangle\}\cup\{\langle 1,j\rangle \mid 1\le j\le 4\}\cup\{2,j \mid 1\le j\le 12\}$,
$v(0,1)=a, v(0,2)=b$,
$v(1,j)=a$ für $1\le j\le 2$, $v(1,j)=b$ für $3\le j\le 4$,
$v(2,j)=a$ für $1\le j\le 6$, $v(2,j)=b$ für $7\le j\le 12$,
$p(0,1)=\langle P_1, a\to a^2\rangle, p(0,2)=\langle P_1, b\to b^2\rangle$,
$p(1,j)=\langle P_2, a\to a^3\rangle$ für $1\le j\le 2$, $p(1,j)=\langle P_2, a\to b^3\rangle$ für $3\le j\le 4$.
D ist eine Ableitung über a^6b^6 von *ab*. Abbildung 2.6 zeigt den Ableitungsbaum.

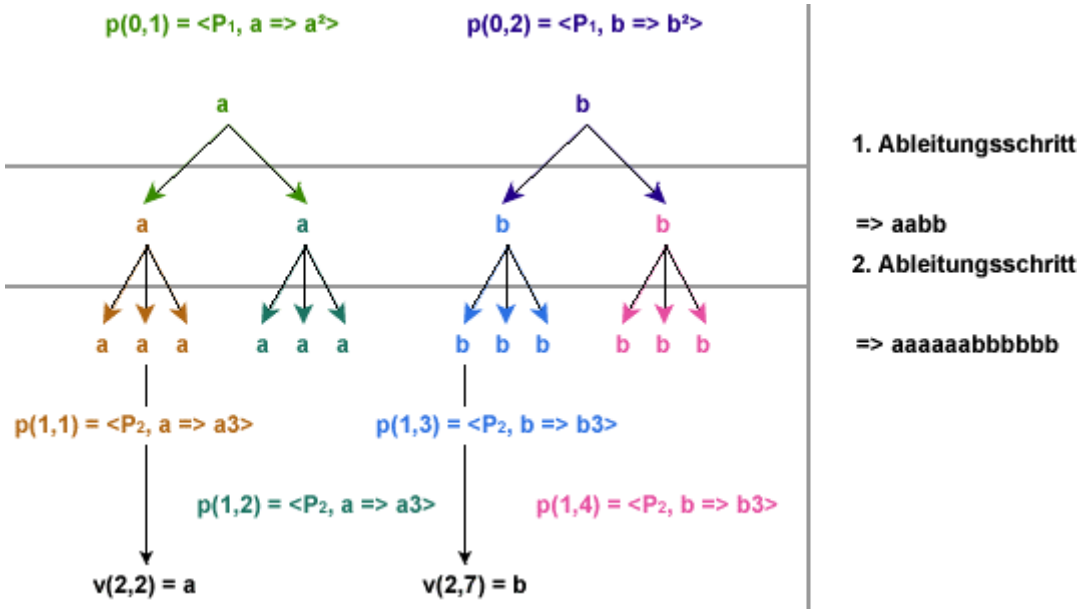


Abbildung 2.6: Ableitungsbaum vom Beispiel 2.8

Definition 2.20 für die Ableitungen eines T0L-Systems [HR75]:
Wenn $G=\langle\Sigma,\mathscr{P},\omega\rangle$ ein T0L-System ist und $S=\langle\Sigma,\mathscr{P}\rangle$ ein T0L-Schema, dann ist eine Ableitung in *S* auch eine *Ableitung in G*. Im Einzelnen, eine Ableitung mit dem trace $x_0,\ldots,x_f$ in *G*, derart das $x_0=\omega$, ist eine *Ableitung über* x_f *in G*.

2.2.3.6 L-Systeme mit Verzweigungen

Schon im „Ur-L-System“ führt Lindenmayer Verzweigungen ein. Bis jetzt wurde darauf noch nicht weiter eingegangen. Das B in diesen B0L-Systemen steht für *branching* oder auch für *bracketed*, weil die eckigen Klammern diese Verzweigungen darstellen. Viele Autoren nehmen die Verzweigungen in ein L-System mit auf und bezeichnen dieses L-System nicht explizit mit einem B. Ein Autor hat in einem Artikel [Ke86] sehr eng mit Lindenmayer zusammengearbeitet, als es um Verzweigungen in L-Systemen ging und diese entsprechend definiert. Daher soll zur Vervollständigung seine Generalisierung mit aufgeführt werden. Dabei stehen die Zeichen „[“ für den Anfang einer Verzweigung und „]“ für das Ende, wie auch in dem ursprünglichen L-System.

Definition 2.21 für ein B0L-Schema [Ke86]:
Ein B0L-Schema $S = \langle \Sigma \cup \{[,]\}, P \rangle$ ist ein 0L-Schema, wenn

1. die Zeichen [,] nicht in Σ enthalten sind,
2. eine Produktion der Form $[\rightarrow [$ und $] \rightarrow]$ existiert
3. und alle Wörter, die über die Produktionen erzeugt werden, wohlgeformte Klammerausdrücke enthalten, falls Klammern enthalten sind.

Definition 2.22 für ein B0L-System [Ke86]:
Ein B0L-System ist ein Tripel $G = \langle \Sigma \cup \{[,]\}, P, \omega \rangle$, wo $S = \langle \Sigma \cup \{[,]\}, P \rangle$ das B0L-Schema ist (auch das *Schema* von *G* genannt) und ω (das *Axiom* von *G*) ist ein Wort über Σ.

Beispiel 2.9 für ein B0L-System:
Folgendes B0L-System sei definiert:

$$
\begin{aligned}
G = (\quad & \Sigma = \{\quad a, \varepsilon \quad\} \cup \{\quad [,] \quad\}, \\
& P = \{\quad a \rightarrow a \,|\, aa \,|\, [a] \,|\, \varepsilon, \\
& \qquad [\rightarrow [, \\
& \qquad] \rightarrow] \qquad\qquad \}, \\
& \omega = \quad a \qquad\qquad\qquad\qquad)
\end{aligned}
$$

Die ersten drei Ableitungsschritte illustriert die Abbildung 2.7.

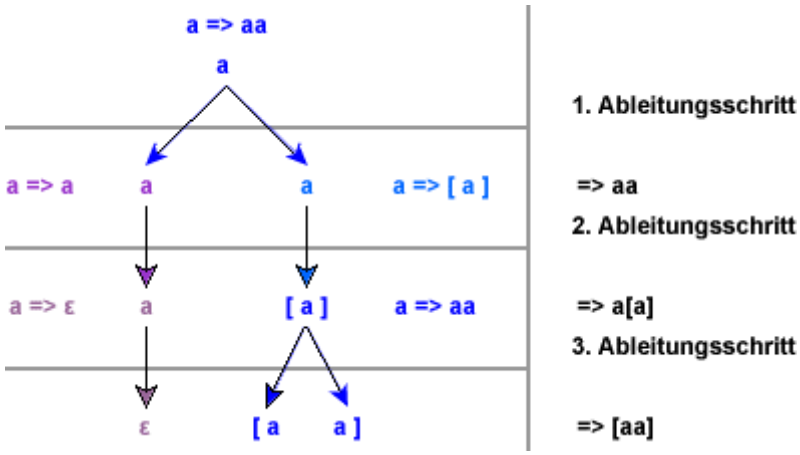


Abbildung 2.7: Ableitungsbaum für das B0L-System

Auswirkungen:
Mit der Einführung von Verzweigungen ist es möglich, nicht nur eindimensionale fadenförmige Organismen, sondern auch zweidimensionale fadenförmige Organismen zu modellieren. Man stelle sich vor, dass die Symbole Zellen darstellen und dabei Stränge bilden. Illustriert wird dies in Abbildung 2.7. Die Auswirkungen sind weitreichend. Es können nun zum Beispiel Pflanzen modelliert werden, aber nur bis zur zweiten Dimension. Das reicht jedoch aus, um die topologische Struktur von Pflanzen zu erzeugen. Es sei darauf hingewiesen, dass noch keine graphische Interpretation des Wortes vorliegt, sondern nur das Wort selbst. Daher ist die Interpretation in Abbildung 2.7 eine von vielen möglichen.

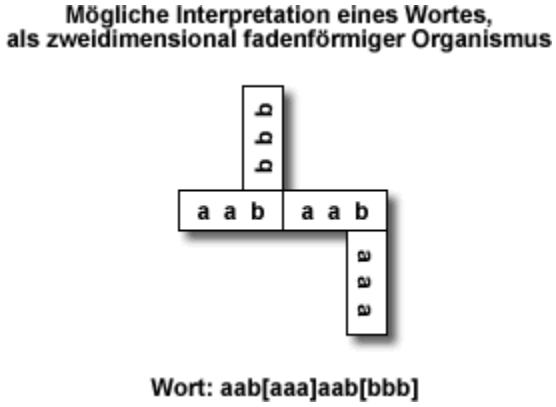


Abbildung 2.8: Zweidimensionaler fadenförmiger Organismus

2.2.3.7 Stochastische L-Systeme

In der Natur ist es fast ausgeschlossen, dass komplexe Strukturen, wie zum Beispiel zwei Bäume, identisch aussehen. Um eine gewisse Zufallskomponente in L-Systeme einzugliedern, wurden die stochastischen 0L-Systeme eingeführt. Daher steht das S auch für *stochastic*. Im frühen Stadium hießen diese L-Systeme *probabilistic* [JM86].
Bei einem „echten“ S0L-System existieren für jedes Symbol mehrere Produktionen. Jede dieser Produktionen bekommt eine Wahrscheinlichkeit von 0 bis 1 zugewiesen, Null selbst ist aber ausgeschlossen, wobei die Summe aller Wahrscheinlichkeiten für alle Produktionen eines Symbols 1 ergeben muss. Somit ist es aus praktischer Sicht ausgeschlossen, dass ein S0L-System deterministisch ist. Ein S0L-System mit nur einer Produktion für jedes Symbol ist gleich einem D0L-System. Die Definition schließt nicht aus, dass S0L-Systeme einem D0L-System entsprechen können.

Definition 2.23 für ein S0L-System [PL90]:
Ein *S0L-System* ist ein geordneter Quadrupel $G = \langle \Sigma, P, \omega, \pi \rangle$. Das Alphabet Σ, das Axiom ω und die Menge der Produktionen P sind definiert wie in einem 0L-System. Die Funktion $\pi : P \to (0,1]$, bezeichnet als *Wahrscheinlichkeitsverteilung*, bildet die Menge der Produktionen auf die Menge der *Produktionswahrscheinlichkeiten* ab. Man kann davon ausgehen, dass für alle Symbole $a \in \Sigma$ die Summe der Wahrscheinlichkeiten aller Produktionen mit der gleichen linken Seite gleich eins ist.

Beispiel 2.10 für stochastische Produktionen:
Es werden Beispiele für Produktionen mit Angabe von deren Wahrscheinlichkeit gegeben.

$$\overset{0.25}{a \to aba}, \qquad \overset{0.25}{a \to a}, \qquad \overset{0.5}{a \to bbba},$$

$$\overset{1}{b \to abc},$$

$$\overset{0.85}{c \to ccba}, \qquad \overset{0.15}{c \to \varepsilon}$$

2.2.3.8 Parametrisierte L-Systeme

Parametrisierte 0L-Systeme bieten die Möglichkeit, bestimmte Regeln nur dann auszuführen, wenn eine bestimmte arithmetische Bedingung erfüllt ist. Dabei muss gesichert sein, dass zu jedem Zeitpunkt eine Ableitung für jedes Symbol in Σ existiert. Damit ist ein „echtes" parametrisiertes L-System nie deterministisch. Wenn es nur eine Produktion für jedes Symbol in Σ gibt, dann muss zwangsweise der Bedingungsteil immer wahr sein. Damit wäre es einem D0L-System gleichzusetzen. Die Definition schließt nicht aus, dass parametrisierte 0L-Systeme einem D0L-System entsprechen können. Mit der Einführung dieser L-Systeme wollten Prusinkiewicz und Lindenmayer beweisen, dass auch komplexe Strukturen mit L-Systemen erzeugt werden können [De03]. Bis dahin waren Aono und Kunii [De03] davon überzeugt, dass nur ihr dreidimensionales prozedurales Modell dazu in der Lage sei, jedoch nicht die L-Systeme.
Formal sieht eine Produktion in einem parametrisierten 0L-System wie folgt aus:

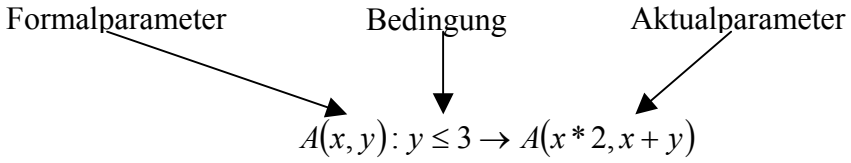


Abbildung 2.9: Struktur einer Produktion eines parametrisierten 0L-Systems

Das Symbol A hat die Formalparameter x und y, denen ein Wert übergeben werden muss. Die Produktion wird aber nur ausgeführt, wenn die Bedingung $y \le 3$ erfüllt ist. Den Symbolen auf der rechten Seite der Produktion werden ein oder mehrere Aktualparameter übergeben, entsprechend der Definition der Formalparameter.

Definition 2.24 für parametrisiertes 0L-System [PL90]:
Ein *parametrisiertes 0L-System* ist definiert als ein geordneter Quadrupel $G = \langle \Sigma, P, \omega, \Lambda \rangle$, wobei gilt

- Σ ist das *Alphabet* des Systems,
- $P \subset (\Sigma \times \Lambda^*) \times C(\Lambda) \times (\Sigma \times E(\Lambda))^*$ ist eine endliche *Menge von Produktionen*,
- $\omega \in (\Sigma \times \Re^*)^+$ ist ein nicht leeres parametrisiertes Wort, genannt *Axiom*,
- Λ ist die Menge von *formalen Parametern*.

In der Definition steht *C(Λ)* (Condition) für die Bedingung in der Produktion und *E(Λ)* (Expression) für den Ausdruck, der eingesetzt wird.

Beispiel 2.11 für ein parametrisiertes P0L-System [PL90]: [2]
Das folgende parametrisierte P0L-System soll diese L-Systeme illustrieren. Dabei ist $\Sigma = \{A, B, C\}$, $\omega = B(2)A(4,4)$ und

$$
\begin{aligned}
P = \{ \quad & A(x,y) && : y \le 3 && \rightarrow A(x*2, x+y), \\
& A(x,y) && : y > 3 && \rightarrow B(x)A(x/y, 0), \\
& B(x) && : x < 1 && \rightarrow C, \\
& B(x) && : x \ge 1 && \rightarrow B(x-1), \\
& C && && \rightarrow C \qquad \}
\end{aligned}
$$

[2] Beispiel in der Quelle ist fehlerhaft.

Die Symbole x und y bilden Variablen, die beim Aufruf der Produktion ihre Werte übergeben bekommen und diese auch als Werte für die Aktualparameter weitergeben können. Es ist eine Voraussetzung von parametrisierten L-Systemen, dass die Variablen auf der rechten Seite für die linke Seite benutzt werden oder für die Bedingung. Es folgt der Ableitungsbaum für die ersten vier Ableitungsschritte.

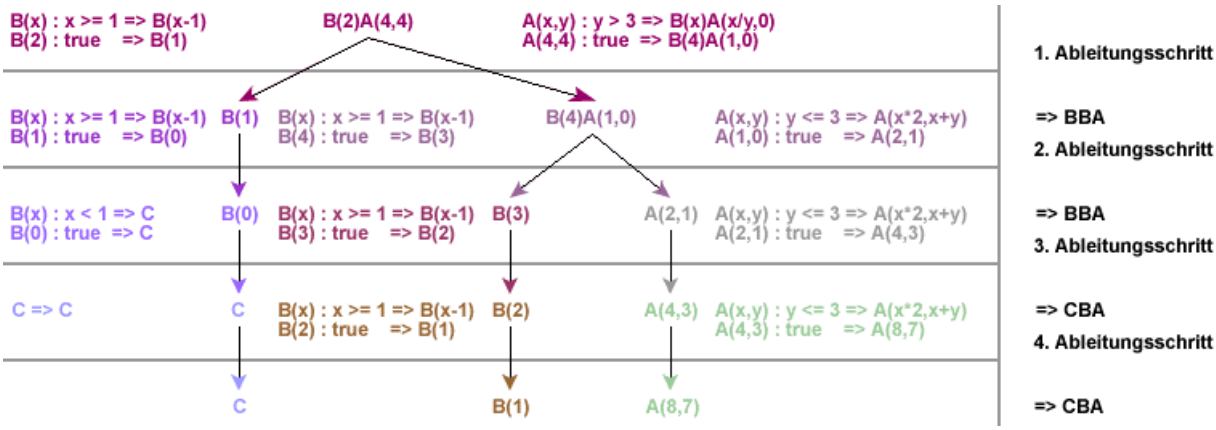


Abbildung 2.10: Ableitungsbaum für das Beispiel 2.11

2.2.3.9 Kontextsensitive L-Systeme

Eine weitere Erweiterung der L-Systeme ist die Kontextsensitivität. Für diese Erweiterung, für ein L-System mit *interaction*, steht das I in dem Begriff IL-System. Statt I kann auch <k, l> stehen, wobei mit k die maximale Sensitivität der linken Seite bezeichnet wird und mit l die der rechten Seite. Alternativ kann für <k, l> auch die Summe von k und l stehen. Wenn zum Beispiel k = 1 und l = 0 ist, dann könnte man <1, 0> L-System schreiben, aber auch 1L-System.
Neben den vielen Möglichkeiten für die Bezeichnung solcher L-Systeme gibt es auch zwei Formen der Bezeichnungen für das eigentliche L-System, im Speziellen für die Produktionen. Diese Unterschiede kommen zustande, weil zwei wichtige Forscher, Rozenberg und Prusinkiewicz, in ihren Büchern und Artikeln zwei verschiedene Notationen benutzen. Weil sich die vorliegende Arbeit mehr in die Richtung Computergrafik bewegt, wird der Notation von Prusinkiewicz der Vorzug gegeben. Der Unterschied liegt in der Beschreibung der Produktionen. Die Produktion eines IL-Systems bei Prusinkiewicz hat die Form [PL90]

$$a_l < a > a_r \rightarrow \alpha .$$

Dabei steht a_l für den Kontext auf der linken Seite und a_r für den Kontext auf der rechten Seite. a_l und a_r können auch eine Länge größer eins besitzen oder nicht vorhanden sein. Ist a_l für eine Produktion nicht vorhanden, kann auch die Schreibweise

$$a > a_r \rightarrow \alpha$$

verwendet werden. Analog gilt für a_r die Schreibweise

$$a_l < a \rightarrow \alpha .$$

Die Produktion eines IL-Systems bei Rozenberg hat die Form [HR75]

$$< a_l , a , a_r > \rightarrow \alpha .$$

Bei einem IL-System ist es auch Voraussetzung, dass zu jedem Zeitpunkt jedes Symbol in Σ abgeleitet werden kann. Zu diesem Zweck können auch kontextfreie Produktionen in solch einem L-System enthalten sein. Sollte zu einem Zeitpunkt für ein Symbol sowohl eine kontextfreie als auch eine kontextsensitive Produktion zutreffen, wird der kontextsensitiven Produktion der Vorzug gegeben.
Die Anwendung für diese L-Systeme ist die Nachbildung des Signalflusses von Organismen. Dadurch ist es möglich, diverses Verhalten so zu beschreiben, dass es Schritt für Schritt verfolgbar ist. Dies ermöglicht es, das Aufblühen einer Blüte oder das Absterben von Pflanzenteilen schrittweise zu modellieren. Dadurch gibt es einen harmonischen Übergang und kein „plötzliches Erscheinen“ mehr.

Beispiel 2.12 für ein PIL-System [PL90]:
Das Beispiel zeigt an einem trivialen PIL-System, wie Signale weitergegeben werden können. Sei $\Sigma = \{a,b\}$, $P = \{b < a \rightarrow b, b \rightarrow a, a \rightarrow a\}$ und $\omega = baaaaaaaa$. Der Ableitungsbaum für die ersten vier Ableitungen zeigt Abbildung 2.10.

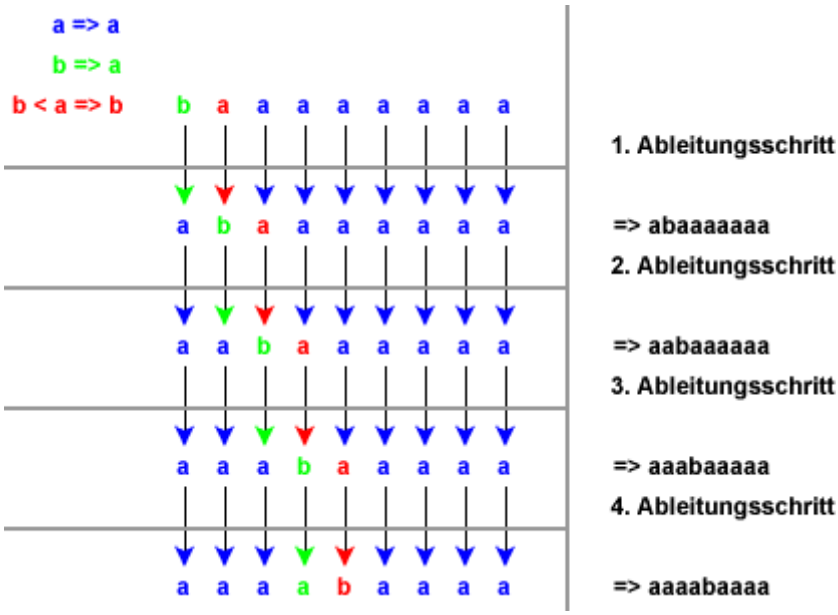


Abbildung 2.11: Signalverlauf in einem PIL-System

2.2.3.10 Weitere L-Systeme

Die bisher besprochenen L-Systeme stellen nur ein Bruchteil der L-Systeme dar, die noch existieren. Einige der noch nicht besprochenen L-Systeme sollen hier lediglich angerissen werden, um die Mächtigkeit und die Vielfalt der L-Systeme zu zeigen.

Map L-System:
Map L-Systeme wurden entwickelt, um die Modellierung von zellularen Ebenen zu ermöglichen. Indem man L-Systeme mit planaren Graphen und Zyklen kombiniert, können komplexe topologische Strukturen von Zellen modelliert werden. Mit Hilfe des Graphen erhält man eine Karte, in der die Kanten Zellwände darstellen und Regionen die Zellen. Dabei gilt, dass keine Inseln innerhalb einer Region existieren dürfen. Kanten selbst können auch Zyklen bilden. [PL90]

Timed L-System:
Aus der Sicht der Computergrafik, interessiert man sich nicht nur für Bilder, sondern auch für Animationen. Sie werden mit diesen „getakteten“ L-Systeme ermöglicht. Dazu werden jedem Symbol Zahlen zugewiesen. Dabei bedeutet die Zahl auf der linken Seite das Sterbealter und die auf der rechten Seite das Initialalter. Beim Ablauf einer Animation wird eine globale Zeitvariable hochgezählt. Erreicht eine Produktion ihr Sterbealter, wird diese Produktion angewendet und fängt mit dem initialen Alter an zu leben. Ein Problem bei diesem Modell ist, dass es kontextfrei ist. Somit kann man zwar das Wachstum animieren, aber nicht mit der Umgebung interagieren. [PL90]

dL-System:.
Das Problem der Timed L-Systeme kann durch ein Hybrid Modell beseitigt werden. Dabei erzeugt das L-System die topologischen Informationen, der eigentliche Animationsprozess wird durch Differentialgleichungen gesteuert. [De03]

Unary L-System:
Das Besondere an diesen L-Systemen ist, dass sie unär sind. In ihrem Alphabet Σ ist lediglich ein Symbol enthalten. Diese L-Systeme finden eher Anwendung im theoretischen Bereich. Sie sind nicht so mächtig wie die anderen L-Systeme, bieten aber die Möglichkeit einer kompletten Charakterisierung der Grammatik. [HR75]

Weitere L-Systeme, die eher theoretischen Charakter haben, sind zum Beispiel A0L-Systeme [KR92] oder Partitions-limitierte L-Systeme [Gä96]. Andere spezielle L-Systeme sind Value L-Systeme oder „forgetful“ L-Systeme [CJ92] [KPM92].
Das eigentliche Problem ist die Bezeichnung von L-Systemen. Es gibt eine große Anzahl von L-Systemen, so dass einige Buchstaben doppelte Bedeutungen aufweisen. Beispielsweise hat Rozenberg das S für symmetrische L-Systeme verwendet. Das S steht aber auch für stochastische L-Systeme.

2.2.3.11 Kombination von L-Systemen

An einigen Beispielen wurde bereits gezeigt, dass sich die L-System-Erweiterungen (T0L, E0L) und Beschränkungen (D0L, P0L) miteinander kombinieren lassen. Dabei sind fast alle erdenklichen Kombinationen möglich, zum Beispiel ET0L-System, PD2L-System oder PSTEIL-System.
Nur sehr wenige L-Systeme lassen sich nicht mit anderen kombinieren, weil es deren Definition nicht zulässt. Ein Beispiel sind die Timed L-Systeme. In ihrer Definition werden diese genauer als Timed D0L-Systeme definiert.
Die Buchstaben der einzelnen L-Systeme müssen nicht in einer bestimmten Ordnung stehen. So kann ein L-System PT0L-System oder auch TP0L-System heißen. Es ist aber üblich, dass die Symbole 0, 1, 2, ... bis I vor dem L stehen. Sollten Tabellen genutzt werden, dann käme als Nächstes ein T links vom Kontext. Wenn Extentions verwendet werden, dann folgt nun ein E links vom T oder links vom Kontext. Links außen steht ein P, wenn es propagierend ist, gefolgt von einem D, wenn es deterministisch ist. Alle anderen Symbole liegen in der Regel zwischen den Bereichen, die von den Zeichen P und D sowie E, T und Kontext gebildet werden.
Abschließend werden die besprochenen L-Systeme noch einmal in einer Übersichtstabelle zusammengefasst.

L-System	Bedeutung	Anwendung
0L-System	Basis Lindenmayer System	Siehe Kapitel 2.1.7
D0L-System | deterministic	Für jedes Symbol existiert nur eine Produktion	Zum automatischen Entwickeln eines Wortes
P0L-System | propagating	ε ist nicht Bestandteil einer Produktion	Entwicklung ohne Absterben von Teilen des Wortes
E0L-System | Extension	Einführung von Terminal- und Nichtterminalzeichen	Schnittstelle zur Chomskyhierarchie
T0L-System | table	Einführung von mehreren Mengen von Produktionen	Verschiedene Umgebungen modellieren (warm, kalt, ...)
B0L-System | branching	Einführung von Verzweigungssymbolen	Mehr dimensionale Strukturen
S0L-System | stochastic	Zuweisung der Produktionen einer Wahrscheinlichkeit für deren Ausführung	Automatische Entwicklung eines Wortes, wobei unterschiedliche Worte generiert werden können
Parametrisierte 0L-System	Erweiterung der Produktionen um eine Bedingung und um Parameter, die übergeben werden können	Realisierung von komplexen Strukturen.
IL-System | interaction	Erweiterung der L-Systeme um Kontextsensivität	Modellierung von Signalverläufen
Map L-System	Graphenbasierte L-Systeme	Zellebenen modellieren
Timed L-System	Produktionen werden um Initial- und Sterbealter erweitert, um eine Animation zu realisieren	Animation eines einzelnen L-Systems
dL-System | differential	Animation einer Szene von L-Systemen durch Differentialgleichungen	Animation mehrerer L-Systeme mit Interaktion ihrer Umgebung
U0L-System | unary	Beschränkung des Alphabetes Σ auf ein Symbol.	Komplette Charakterisierung der L-Systeme

Tabelle 2.3: Übersichtstabelle der besprochenen L-Systeme

2.2.4 Wachstumsfunktion

Im Beispiel 2.1 wurde demonstriert, dass das Wort eines L-Systems schnell anwachsen kann. Um dieses Wachstum mathematisch zu beschreiben, wurden Wachstumsfunktionen eingeführt. Die Wachstumsfunktion ist eine Funktion, die zu jedem Zeitpunkt die Anzahl der Symbole eines Wortes wiedergeben kann.
Bei einem D0L-System ist zu beobachten, dass die Wachstumsfunktion nicht abhängig von der Anordnung der Symbole im Wort ist und auch nicht von den abgeleiteten Wörtern ist. Daher ergibt sich eine Relation zwischen dem Vorkommen der Anzahl der Symbole und den zwei Wörtern μ und υ, wobei μ das abzuleitende Wort ist und υ das abgeleitete Wort darstellt. Diese Relation lässt sich durch eine Matrix beschreiben.

Es sei ein D0L-System gegeben, dessen Alphabet Σ geordnet ist, $\Sigma = \{a_1,...,a_m\}$, dabei ist *m* eine nicht negative ganze Zahl, die größer Null ist. Die Matrix $Q_{m\times m}$ ist so aufgebaut, dass jedes Element $q_{i,j}$ das Vorkommen eines Symbols a_j auf der rechten Seite einer Produktion a_i enthält. Sei a_i^k die Anzahl des Vorkommens eines Symbols a_i in einem generierten Wort *x* und *k* die Anzahl der entsprechenden Ableitungsschritte. Es lässt sich nun eine Matrix für die direkte Ableitung eines Wortes auf der Grundlage eines D0L-Systems erzeugen. [PL90]

$$\begin{bmatrix} a_1^k & \cdots & a_m^k \end{bmatrix} \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1m} \\ q_{21} & q_{22} & \cdots & q_{2m} \\ \vdots & & & \\ q_{m1} & q_{m2} & \cdots & q_{mn} \end{bmatrix} = \begin{bmatrix} a_1^{k+1} & \cdots & a_m^{k+1} \end{bmatrix}$$

Beispiel 2.13 für die Wachstumsfunktion [PL90]:
Es sei folgendes PD0L-System gegeben.

$$\begin{aligned} G = (\ & \Sigma = \{\ a,b\ \}, \\ & P = \{\ a \to ab, \\ & \qquad b \to a\ \}, \\ & \omega = \ a\ ) \end{aligned}$$

Die entsprechende Matrix lautet

$$\begin{bmatrix} a^k & b^k \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} a^{k+1} & b^{k+1} \end{bmatrix}$$

oder alternativ

$$a^{k+1} = a^k + b^k = a^k + a^{k-1}.$$

Lässt man nun einige Ableitungsschritte durchführen (für k = 1, 2, 3, ...), erhält man für die Wachstumsfunktion die Fibonacci-Reihe: 1, 1, 2, 3, 5, 8, ...

Rozenberg und Salomaa zeigen in „The Mathematical Theory of L Systems“ [PL90], dass die Wachstumsfunktion eines jeden D0L-Systems eine Kombination aus einer exponentialen und ganz rationalen Funktion ist.
Aus der Sicht der realistischen Pflanzenmodellierung sind solche Wachstumsfunktionen nicht verwendbar, weil eine Pflanze nicht exponentiell wächst. Um das Pflanzenwachstum realistisch zu modellieren, wäre daher die Nutzung einer Sigmoidfunktion für D0L-Systeme oder die Nutzung der Funktion einer Quadratwurzel für IL-Systeme angebracht. [PL90]

In diesem Zusammenhang sei noch erwähnt, dass Wachstumsfunktionen einen eher theoretischen Charakter haben. In der Praxis ist es nur sehr schwer möglich, eine Wachstumsfunktion für ein gegebenes L-System zu finden. Eine differenzierte Darstellung der Materie ist zu finden bei. [HR75]

2.2.5 Klassifikation / Einordnung

L-Systeme in Verbindung zu setzen oder zusammenzubringen ist keine leichte Aufgabe. Weil der Schwerpunkt dieser Arbeit nicht auf der Klassifikation von L-Systemen liegt, werden die L-Systeme größtenteils in einen bestimmten Kontext eingeordnet. Da die Grafiken sehr groß sind, werden sie ausgegliedert und auf ein Poster gedruckt. Wie alle anderen Abbildungen liegen aber auch sie auf der CD-ROM vor.

Zeitstrahl:
Die wichtigsten besprochenen L-Systeme werden auf einen Zeitstrahl gesetzt und chronologisch nach dem Datum ihrer Veröffentlichung eingeordnet. Dabei wird zu jedem L-System auch die entsprechende Quelle angegeben. Der Zeitstrahl fängt beim ursprünglichen mathematischen Modell im Jahre 1968 an und endet 1993 mit den differentiellen L-Systemen zur Realisierung von Animationen. Seit 1993 sind keine weiteren wichtigen L-Systeme dazugekommen, bzw. bestehende unbedeutende haben sich nicht zu bedeutsamen L-Systemen qualifiziert [HR75] [PL90] [De03] [RS86].

Eigenschaften von L-Systemen:
Die verschiedenen Eigenschaften von L-Systemen lassen sich stark vereinfacht in zwei Klassen einordnen.
In die erste Klasse lassen sich die Beschränkungen einordnen. Basierend auf einem Ausgangssystem, in diesem Fall das 0L-System, wird untersucht, welche Beschränkungen für dieses System vorliegen, beispielsweise ob es deterministisch ist oder propagierend, wobei die beschränkten L-Systeme eine echte Teilmenge zum Ausgangssystem bilden [HR75]. Je gravierender die Beschränkung ist, desto weiter weg steht das L-System vom Ausgangssystem. Da einem P0L-System nur ein Zeichen fehlt, bildet es keine große Beschränkung. Eine größere Beschränkung ist dagegen, dass für jedes Symbol in Σ nur eine Produktion existieren darf. Die größte Beschränkung ist jedoch, das Σ nur ein Symbol enthalten darf, wie bei den U0L-Systemen. All diese L-Systeme stellen eine beschränkte Eigenschaft dar und reduzieren die Mächtigkeiten der Sprache.
In die zweite Klasse lassen sich die Erweiterungen der L-Systeme einordnen. Je weiter weg eine Erweiterung ist, desto größer ist die Leistung, die diese Erweiterung mit sich bringt. Das einfache getrennte Hinzufügen von zwei Klammern bei den B0L-Systemen stellt dabei die niedrigste Erweiterung dar. Die Klammern sind zwar für die graphische Interpretation wichtig, aber nicht für die reine Mächtigkeit der Sprache. Timed L-Systeme und S0L-Systeme bilden eine größere Erweiterung zum Ausgangssystem. S0L-Systeme ermöglichen eine kontrollierte automatische Ableitung von nicht deterministischen Produktionen, während Timed L-Systeme den Produktionen ein Initial- und Sterbealter zuweisen, um festzulegen, wann diese angewandt werden. Die mächtigsten Erweiterungen bilden die T0L, IL, E0L, dL und die parametrisierten L-Systeme. Mit den T0L-Systemen ist die Modellierung von mehreren Umgebungen möglich. Durch IL-Systeme lassen sich Signalflüsse modellieren. Die E0L-Grammatik führt ein zweites Alphabet ein, das der Terminale. Die dL-Systeme bieten die Möglichkeit der Animation von ganzen Szenarien, bzw. das kontrollierte Wachstum solcher Szenarien. Mit parametrisierten L-Systemen ist die Modellierung sämtlicher komplexer Objekte möglich. Alle diese L-Systeme erweiterten das 0L-System um zusätzliche Mächtigkeit.

Schnittstellen zur Chomskyhierarchie:
Hierfür wurde auf [HR75] zurückgegriffen, Theorem 10.10. Dabei stellt eine gerichtete Kante dar, dass die eine Sprachfamilie in der anderen enthalten ist. Wenn zwei Sprachfamilien durch einen Pfad in diesem gerichteten Graphen nicht erreichbar sind, sind diese inkompatibel aber nicht disjunkt. Dabei ergibt sich Folgendes:

- Klassen sind nicht disjunkt (Lemma 10.1 [HR75]).
- $F(RG) \subset F(CF)$ (Theorem1.1 [HR75]),
  $F(0L) \subset F(T0L)$ (Lemma 10.2 [HR75]),
  $F(0L) \subset F(IL)$ (Lemma 10.2 [HR75]),
  $F(0L) \subset F(E0L)$ (Lemma 10.2 [HR75]),
  $F(CF) \subset F(E0L)$ (Corollary 10.1 [HR75]),
  $F(T0L) \subset F(ET0L)$ (Lemma 10.2 [HR75]),

$F(IL) \subset F(EIL)$ (Lemma 10.2 [HR75]),
$F(E0L) \subset F(ET0L)$ (Lemma 10.2 [HR75]),
$F(ET0L) \subset F(CS)$ (Theorem 10.5 [HR75]),
$F(CS) \subset F(RE)$ (Theorem 1.1 [HR75]).

- $F(EIL) \subset F(RE)$ (Theorem 10.6 [HR75]).
- $F(RG) \subset F(T0L)$ (Lemma 10.1 [HR75]),

$F(RG) \subset F(IL)$ (Lemma 10.1 [HR75]),
$F(0L) \subset F(CF)$ (Theorem 10.4 [HR75]),
$F(T0L) \subset F(IL)$ (Theorem 10.9 [HR75]),
$F(T0L) \subset F(E0L)$ (Theorem 10.8 [HR75]),
$F(IL) \subset F(CS)$ (Theorem 10.7 [HR75]).

Entscheidungsautomat:
Die wichtigsten L-Systeme für die Computergrafik werden zu einem Automaten zusammengefügt. Der Automat erzeugt eine Zeichenkette, die ein L-System darstellt, das die Eigenschaften aufweist, die zur Modellierung benötigt werden. Dazu wird in jedem Zustand eine Frage gestellt, die mit „J“ (Ja) oder „N“ (Nein) beantwortet werden muss. Zu jeder Frage gibt es ein Beispiel, das die Frage verständlicher machen soll.

2.2.6 Interpretation des Wortes

1986 stellt Prusinkiewicz [Pr86] eine Möglichkeit vor, wie man das Wort eines L-Systems graphisch interpretieren könnte. Dabei zeigt er, wie man die Turtle-Interpretation für Wörter der L-Systeme nutzen könnte. Szilard und Quinton zeigen schon 1979, wie die Turtle-Interpretation auf D0L-Systemen genutzt werden könnte, aber erst Prusinkiewicz verallgemeinerte dies. [Ja97] Hinter dieser Entwicklung steht die Idee, das Verhalten einer Schildkröte bei der Bewegung zu imitieren (sie läuft so lange geradeaus ohne die Richtung zu ändern, bis sie gestört wird oder durch ein anderes Bedürfnis dazu veranlasst wird).
Prusinkiewicz definiert die Turtle-Interpretation wie folgt.

Definition 2.25 für die Turtle-Kommandos im 2D-Raum [Pr86]:
Der Zustand einer Schildkröte ist ein Tripel *(x, y, α)*, wobei die Koordinaten *(x, y)*, die Position der Schildkröte repräsentieren und der Winkel *α*, auch Ausrichtung der Schildkröte genannt, die Richtung angibt, in die die Schildkröte blickt. Des Weiteren soll *d* die Schrittweite eines Schrittes sein und *δ* die Schrittweite, die bei einer Veränderung des Winkels *α* benutzt wird. Es seien folgende Kommandos für die Schildkröte definiert:

F Die Schildkröte bewegt sich um die Schrittweite *d* vorwärts und zeichnet eine Gerade über die Strecke, die sie zurücklegt. Dabei verändert sich der Zustand der Schildkröte wie folgt:
$$(x, y, \alpha) \to (x + d\cos\alpha, y + d\sin\alpha, \alpha)$$

f Die Schildkröte bewegt sich um die Schrittweite d vorwärts und zeichnet keine Gerade. Dabei verändert sich der Zustand der Schildkröte wie folgt:
$$(x, y, \alpha) \to (x + d\cos\alpha, y + d\sin\alpha, \alpha)$$

\+ Die Richtung der Schildkröte wird um die Schrittweite *δ* weiter nach links ausgerichtet. Der Zustand der Schildkröte verändert sich wie folgt:
$$(x, y, \alpha) \to (x, y, \alpha + \delta)$$

- Die Richtung der Schildkröte wird um die Schrittweite δ weiter nach rechts ausgerichtet. Der Zustand der Schildkröte verändert sich wie folgt:
  $(x, y, \alpha) \rightarrow (x, y, \alpha - \delta)$

Alle anderen Symbole werden ignoriert.

Definition 2.26 für die Turtle-Interpretation im 2D-Raum [Pr86]:
Sei *v* eine Zeichenkette, (x_0, y_0, α_0) der Startzustand der Schildkröte sowie *d und* δ feste Parameter. Basierend auf die Kommandos, die in der Zeichenkette *v* enthalten sind, wird das Bild von der Schildkröte gezeichnet. Dies nennt man *Turtle-Interpretation* von *v* im zweidimensionalen Raum.

Beispiel 2.14 für die quadratische Koch Insel [Pr86]:
Es sei folgendes PD0L-System definiert.

$$\begin{aligned} G = (\quad & \Sigma = \{ \quad F, -, + \quad \}, \\ & P = \quad F \rightarrow F + F - F - FF + F + F - F, \\ & \omega = \quad F + F + F + F \quad) \end{aligned}$$

Des Weiteren sei $\delta = 90°$. Es folgt die *Turtle-Interpretation* für die ersten drei generierten Worte. Sie bildet die quadratische Koch Insel.

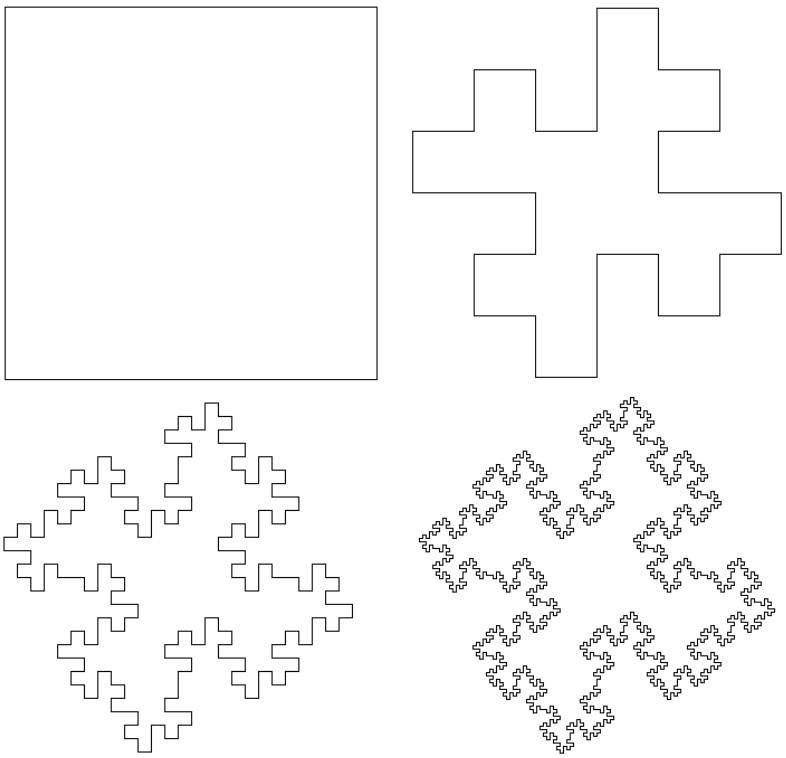


Abbildung 2.12: Die graphische Interpretation der ersten vier generierten Worte

Um nicht nur eine starre Folge von aneinander gereihten Linien zu erhalten, werden zur Turtle-Interpretation zwei weitere Symbole hinzugefügt. Diese Symbole helfen Verzweigungsstrukturen miteinzubeziehen. In Anlehnung an Lindenmayer werden dazu die Symbole „[“ und „]“ hinzugenommen.

[Lege den aktuellen Zustand der Schildkröte auf den Stack. Es können auch optional weitere Attribute (wie die Farbe) oder Parameter (wie *d*) mit abgelegt werden.

] Hole einen Zustand vom Stack und setze die Schildkröte an die Position, die der Zustand beschreibt. Zeichne beim Umsetzen der Schildkröte keine Linie. Sollten weitere Attribute vorliegen (wie Farbe) oder Parameter (wie *d*), dann werden die derzeitigen Attribute und Parameter mit den neuen aktualisiert.

Sollte ein Zustand vom Stack geholt werden und dieser ist leer, dann gibt es für das Wort υ keine Interpretation. In diesem Fall wird eine Fehlermeldung ausgegeben.

Beispiel 2.15 für eine einfache Pflanze [Pr86]:
Durch die Bedeutung der Symbole lassen sich zum Beispiel Pflanzen beschreiben. Die graphische Interpretation dieses Beispiels soll eine Pflanze darstellen. Gegeben sei folgendes PDB0L-System.

$$
\begin{aligned}
G = (\quad & (\Sigma = \{\quad F, X, -, +\}) \cup \{[,]\}, \\
& P = \{\quad X \quad \rightarrow F - [[X] + X] + F[+FX] - X, \\
& \qquad F \quad \rightarrow FF, \\
& \qquad [\quad \rightarrow [, \\
& \qquad] \quad \rightarrow] \qquad\qquad \}, \\
& \omega = X \qquad\qquad\qquad\qquad)
\end{aligned}
$$

Des Weiteren sei $\delta = 22{,}5°$. Die folgenden Abbildungen zeigen die Ergebnisse nach drei, vier, fünf und sechs Ableitungen. Die Bilder aus den Beispielen 2.14 und 2.15 wurden mit dem Applet LSys.Class generiert. [[Lap]]

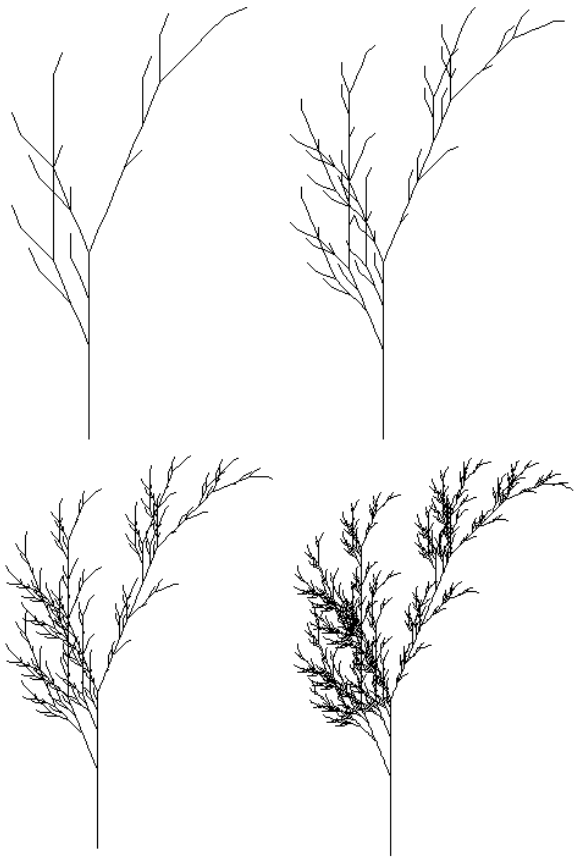


Abbildung 2.13: Die Entwicklung einer Pflanze mit einem L-System

Eine zweidimensionale Grafik, welche die topologische Struktur der Pflanze visualisiert, reicht nicht aus, um realistische Pflanzen zu modellieren. . Hierzu muss die *Turtle-Interpretation* in den 3D-Raum überführt werden.
Zu diesem Zweck wird zu der x- und y-Koordinate noch die z-Koordinate hinzugefügt sowie eine Rotationsmatrix *M*. Mit Hilfe der Rotationsmatrix kann sich die Schildkröte um jede Achse drehen, dazu muss nur die Rotationsmatrix mit einer entsprechenden Matrix verknüpft werden. Die Abbildung 2.13 verdeutlicht dies.

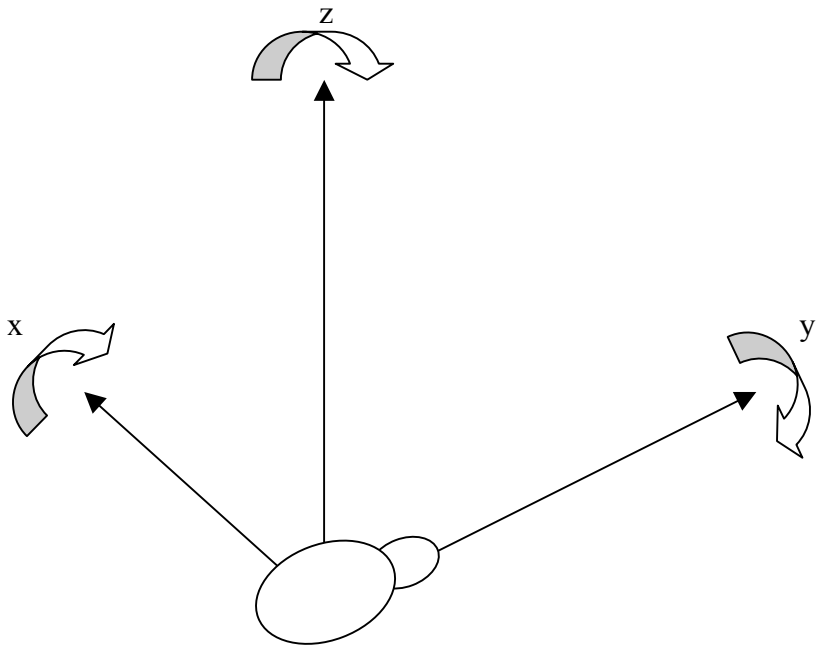


Abbildung 2.14: Turtle-Interpretation im dreidimensionalen Raum

Definition 2.27 für die Turtle-Kommandos im 3D-Raum [De03]:
Der Zustand einer Schildkröte ist ein Quadrupel (x, y, z, M), wobei die Koordinaten (x, y, z) die Position der Schildkröte repräsentieren. M ist die Rotationsmatrix, die die Ausrichtung der Schildkröte im dreidimensionalen Raum beschreibt, also wohin die Schildkröte blickt. Des Weiteren soll $\vec{d}$ die Schrittweite eines Schrittes sein und δ die Schrittweite, die bei einer Veränderung der Rotationsmatrix *M* benutzt wird. Außerdem seien folgende Rotationsmatrizen definiert, die angewandt werden, wenn auf einer entsprechenden Achse rotiert wird.

$$R_x(\delta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\delta & -\sin\delta \\ 0 & \sin\delta & \cos\delta \end{pmatrix}$$

$$R_y(\delta) = \begin{pmatrix} \cos\delta & 0 & -\sin\delta \\ 0 & 1 & 0 \\ \sin\delta & 0 & \cos\delta \end{pmatrix}$$

$$R_z(\delta) = \begin{pmatrix} \cos\delta & \sin\delta & 0 \\ -\sin\delta & \cos\delta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Es seien folgende Kommandos für die Schildkröte definiert:

F Die Schildkröte bewegt sich um die Schrittweite *d* vorwärts und zeichnet eine Gerade über die Strecke, die sie zurücklegt. Dabei verändert sich der Zustand der Schildkröte wie folgt:

$$(x, y, z, M) \to \left(x + \left(M \vec{d} \right)_1, y + \left(M \vec{d} \right)_2, z + \left(M \vec{d} \right)_3, M \right)$$

f Die Schildkröte bewegt sich um die Schrittweite d vorwärts und zeichnet keine Gerade. Dabei verändert sich der Zustand der Schildkröte wie folgt:

$$(x, y, z, M) \to \left(x + \left(M \vec{d} \right)_1, y + \left(M \vec{d} \right)_2, z + \left(M \vec{d} \right)_3, M \right)$$

\+ rotiert die Schildkröte um die Schrittweite δ auf der y-Achse. Der Zustand der Schildkröte verändert sich wie folgt:

$$(x, y, z, M) \to (x, y, z, M \cdot R_y(\delta))$$

\- rotiert die Schildkröte um die Schrittweite -δ auf der y-Achse. Der Zustand der Schildkröte verändert sich wie folgt:

$$(x, y, z, M) \to (x, y, z, M \cdot R_y(-\delta))$$

& rotiert die Schildkröte um die Schrittweite δ auf der x-Achse. Der Zustand der Schildkröte verändert sich wie folgt:

$$(x, y, z, M) \to (x, y, z, M \cdot R_x(\delta))$$

^ rotiert die Schildkröte um die Schrittweite -δ auf der x-Achse. Der Zustand der Schildkröte verändert sich wie folgt:

$$(x, y, z, M) \to (x, y, z, M \cdot R_x(-\delta))$$

\ rotiert die Schildkröte um die Schrittweite δ auf der z-Achse. Der Zustand der Schildkröte verändert sich wie folgt:

$$(x, y, z, M) \to (x, y, z, M \cdot R_z(\delta))$$

/ rotiert die Schildkröte um die Schrittweite -δ auf der z-Achse. Der Zustand der Schildkröte verändert sich wie folgt:

$$(x, y, z, M) \to (x, y, z, M \cdot R_z(-\delta))$$

| dreht die Schildkröte um 180° um die y-Achse. Der Zustand der Schildkröte verändert sich wie folgt:

$$(x, y, z, M) \to (x, y, z, M \cdot R_y(180°))$$

Alle anderen Symbole werden ignoriert.
Die eckigen Klammern behalten ihre Funktion bei und verändern sich nicht bei der *Turtle-Interpretation* im dreidimensionalen Raum.

Definition 2.28 für die Turtle-Interpretation im 3D-Raum:
Sei *v* eine Zeichenkette, (x_0, y_0, z_0, M_0) der Startzustand der Schildkröte sowie *d* und δ feste Parameter. Das Bild, das von der Schildkröte auf der Basis der Kommandos gezeichnet wurde, die in der Zeichenkette *v* enthalten sind, nennt man *Turtle-Interpretation* von *v* im dreidimensionalen Raum.

Im Rahmen der Erläuterung der L-Systeme wurden bereits parametrisierte L-Systeme vorgestellt. Diese Parametrisierung kann auch auf die Kommandos der Schildkröte ausgeweitet werden. Da es nur zwei Arten von Kommandos gibt, bewegen und rotieren, soll exemplarisch für jeweils eines dieser beiden Kommandos die parametrisierte Form angegeben werden. Für die anderen ist es dann analog umzusetzen.

F Die Schildkröte bewegt sich um die Schrittweite *d*, welche einen Parameter *w* übergeben bekommt, vorwärts und zeichnet eine Gerade über die Strecke, die sie zurücklegt. Dabei verändert sich der Zustand der Schildkröte wie folgt:

$$(x, y, z, M) \to \left(x + \left(M \vec{d}(w) \right)_1, y + \left(M \vec{d}(w) \right)_2, z + \left(M \vec{d}(w) \right)_3, M \right)$$

\+ rotiert die Schildkröte um die Schrittweite δ, die einen Parameter *w* übergeben bekommt, auf der y-Achse. Der Zustand der Schildkröte verändert sich wie folgt:

$$(x, y, z, M) \to (x, y, z, M \cdot R_y(w))$$

w kann dabei ein Wert aus dem Bereich der reellen Zahlen sein, entsprechend dem Wert in einem parametrisierten L-System.

Neben diesen Standardkommandos gibt es noch einige zusätzliche Kommandos, die zur realistischen Modellierung von Pflanzen hinzugefügt werden. [PL90] [PHM]
So gelten die Punkte in den geschweiften Klammern ({ }) als Punkte für ein Polygon, das bei der graphischen Interpretation eine geschlossen Fläche darstellt. Damit lassen sich zum Beispiel Blätter darstellen. Mit dem Ausrufezeichen (!) kann der Astdurchmesser vermindert werden und mit dem ' Zeichen kann die Farbe gesteuert werden.
Es gibt aber keinen standardisierten Satz an Kommandos. Daher können weitere Kommandos, die nicht zum besprochenen Vorrat der Kommandos zählen, in Applikationen verwendet werden. Im folgenden Beispiel werden einige dieser proprioritären Kommandos benutzt. Die Applikation, die diese 3D-Grafik erstellt hat, heißt L-System 4 und ist auf der CD-ROM mit enthalten. Zur Interpretation dieser zusätzlichen Kommandos sei auf die Hilfe-Datei verwiesen.

Beispiel 2.16 für einen mit L-System 4 [[LS4]] erzeugten Baum:
Es sei folgendes parametrisiertes PDB0L-System gegeben, welches eine 3D-Grafik erzeugt, die in Abbildung 2.14 zu sehen ist.

$$\begin{aligned}
G = (\ & (\Sigma = \{A, B, C, D, F, L, T, c, f, g, t, z, \&, \$, '\sim, _, >, -, +\}) \cup \{[,]\}, \\
& P = \{\ T \to CCA, \\
& \quad A \to CBD > (94)CBD > (132)BD, \\
& \quad B \to [\&CDCD\$A], \\
& \quad D \to [g(50)Lg(50)Lg(50)Lg(50)Lg(50)Lg(50)L], \\
& \quad C \to !(.95)\sim(5)tF, \\
& \quad F \to '(1.25)F'(.8), \\
& \quad L \to [\sim f(200)c(8)\{+(30)f(200)\text{-}(120)f(200)\text{-}(120)f(200)\}], \\
& \quad f \to z, \\
& \quad z \to _, \\
& \quad [\ \to [, \\
& \quad]\ \to] \qquad \}, \\
& \omega = c(12)T \qquad)
\end{aligned}$$

Des Weiteren soll δ = 20° sein und der Astdurchmesser = 50. Für die Abbildung 2.14 wurden 15 Ableitungsschritte durchgeführt. Das Wort enthält 1.337.641 Zeichen und 17.494 3D-Primitive.

Abbildung 2.15: Die 3D-Grafik zum Beispiel 2.16

Anzumerken ist, dass es auch alternative graphische Interpretationen gibt. Diese haben sich gegenüber der *Turtle-Interpretation* jedoch nicht durchsetzten können, vergleiche dazu [HR75].

2.2.7 Übersicht über Anwendungsmöglichkeiten von L-Systemen

Bisher wurden zahlreiche L-Systeme sowie ihre Interpretation vorgestellt. Auch einige Stichworte zu den Anwendungsmöglichkeiten von verschiedenen L-Systemen wurden bereits genannt. Dieses Kapitel stellt einige weitere Anwendungsmöglichkeiten vor. Dabei werden die Anwendungen in drei Kategorien eingeteilt.

Computergrafik:

- *Pflanzenmodellierung*
  Durch die Arbeiten von Prusinkiewicz ist die Pflanzenmodellierung einer der Hauptanwendungsbereiche für L-Systeme [PL90].
- *Modellierung auf zellularer Ebene*
  Prusinkiewicz [PL90] nutzte L-Systeme auch zur Visualisierung ihrer ursprünglichen Funktion. Dazu kombiniert er Map L-Systeme, die die topologische Struktur beschreiben, mit einer graphischen Interpretation, die auf [NLA86] basiert. Hierbei ist das Axiom ein reguläres Polynom, das die Anfangskarte mit deren Grenzen enthält. Die Zellwand ist eine Linie, entsprechend den Kanten des Graphen. Wenn eine Wand durch eine Produktion ge-

teilt wird, dann werden alle Folgewände auf die gleiche Länge ausgerichtet. Die Position einer Wand basiert auf der Position zweier passender Marker.

- *Fraktale*
  Eine weitere Anwendung finden L-Systeme in der Beschreibung und graphischen Interpretation von Fraktalen. So erzeugt ein L-System mit dem Axiom $\omega = F\text{-}F\text{-}F\text{-}F$ und der Produktion $P = F \rightarrow F - F + F + FF - F - F + F$ eine quadratische Koch Insel. Dabei wird ein Winkel von $\delta = 90°$ angenommen [PL90]. Die Arterien eines Menschen sind ebenfalls fraktalartig aufgebaut. In [Za01] werden Arterien mit L-Systemen modelliert.
- *Gebirge und Gegenstände*
  ELSYS ist ein Programm, das für Sun Workstations entwickelt wurde. Es enthält einen L-Compiler, ein Grafikmodul und ein User Interface. Der L-Compiler übersetzt eine Sprache in eine Grafiksequenz für das Grafikmodul. Dabei besteht die Sprache aus einer Mischung aus Hochsprache, L-Systemen und Erweiterungen für L-Systeme. So können Gebirge auf der Basis von Fraktalen erzeugt werden, die mit L-Systemen beschrieben wurden. [GR92]
- *Kreaturen und Gegenstände*
  Hornby nutzt L-Systeme in Verbindung mit den Genetischen Algorithmen. Dabei wird eine Initialpopulation erzeugt, die aus Produktionen von L-Systemen besteht. Dieser Population wird mit Hilfe von Genetischen Algorithmen und deren Operatoren verändert. So werden, neben Kreaturen, Gegenstände wie Tische oder Bauanleitungen auf der 2D-Ebene für Roboter entwickelt. [HLP01] [HP01a] [HP01b] [HP01c] [HP02]
- *Städtebau*
  Das Programm City-Engine wird zur kompletten Entwicklung von Städten genutzt, wobei L-Systeme zur Beschreibung von vollständigen Straßenzügen und allen Gebäuden verwendet werden [Bü03].
- *Federn*
  Federn weisen eine ähnliche Struktur wie Blätter auf. Deshalb kann vom Wissen der Pflanzenmodellierung profitiert werden, um auch Federn zu modellieren. [Gw03]

Theoretische Informatik:

- *Einordnung in die Chomskyhierarchie*
  Rozenberg zeigt in [HR75], wie mit Hilfe der E0L-Systeme eine Verbindung zur Chomskyhierarchie aufgebaut werden kann. Weiterhin wird versucht, wie in [Gä96], L-Systeme so weit wie möglich in die Chomskyhierarchie einzugliedern. Weitere Quelle zum Thema. [Wa01]
- *Komplexität von L-Systemen*
  Für komplexere L-Systeme ist es schwer, wenn nicht unmöglich, die Berechnungskomplexität zu bestimmen. Wie schon im Kapitel über die Wachstumsfunktionen erwähnt, haben Rozenberg und Salomaa in *„The Mathematical Theory of L Systems“* gezeigt, dass die Wachstumsfunktion von D0L-Systemen allgemein eine Kombination aus exponentieller und polynominaler Funktion ist. Weitere Quellen zur Komplexität von L-Systemen. [LS92] [Ke86]
- *Zugehörigkeitsproblem*
  Eine zentrale Frage im Bereich der Formalen Sprachen ist, ob ein willkürliches Wort zur Sprache einer Grammatik gehört. Für grundlegende 0L-Systeme wird das Problem in [HR75] behandelt.
- *Äquivalenzproblem*
  Eine weitere Forschungsrichtung in diesem Bereich setzt sich mit der Frage auseinander, ob zwei willkürliche Grammatiken von L-Systemen dieselbe Sprache erzeugen. Auch hier hat Rozenberg in seinem Buch [HR75] die Grundlagen für typische L-Systeme gelegt. Weitere Quellen zum Äquivalenzproblem. [Ru86] [CK86] [Hon01] [Wa01]

Sonstige Anwendungen:

- *Routing Tabellen für Computernetze*
  In [LT86] wird gezeigt, wie man das Problem der anwachsenden Routing Tabellen alternativ lösen kann. Umsetzungen lassen sich über Table L-Systeme oder auch mit Map L-Systemen realisieren.
- *Parallele Kommunikationssysteme*
  In [Pa92] wird gezeigt, dass L-Systeme auch die Rolle eines parallelen Kommunikationssystems annehmen können.
- *Modellierung der Netzhaut*
  GREDEA ist ein Monitor-Programm zur Prüfung der Blutzirkulation der Netzhaut. Dabei werden zur Beschreibung der Netzhaut parametrisierte L-Systeme verwendet. [KTV99a] [KTV99b]
- *Erzeugung von Musik*
  Tom Johnson nutzte L-Systeme zur Generierung von Musikstücken. Dabei wird ein L-System erzeugt und n mal abgeleitet. Jedes erzeugte Wort bildet eine Stimme. Das erste Wort die tiefste und das letzte erzeugte Wort die höchste Stimme. Alle Stimmen werden auf ein Tempo befördert, indem der Tonwechsel bei der ersten Ableitung am langsamsten und bei der letzten Ableitung am schnellsten ist [Be01]. Informationen zur Erzeugung von Musik mit L-Systemen als Beschreibung und mit Genetischen Algorithmen zur Generierung findet man bei [Fo00]. Andere Anwendungen in der Musik findet man in [DB03].
- *Evolution von Neuronalen Netzen*
  In [Sc02] werden einfache Beispiele dafür gezeigt, wie die Beschreibung von L-Systemen genutzt werden kann, um Neuronale Netze wachsen zu lassen. So kann mit Genetischen Algorithmen die Produktionen eines L-Systems verändert werden und anschließend ein Wort generiert werden, welches als Neuronales Netz interpretiert werden kann.
- *Analyse von Gehölzen*
  L-Systeme können aber auch zur Analyse von Gehölzen benutzt werden. So können der Saftfluss oder die Auswirkungen von Parasiten und Insekten auf die Holzqualität verfolgt werden. Spezialist auf diesem Gebiet ist Prof. Dr. Winfried Kurth von der BTU Cottbus. In seiner Monografie [Ku99] geht er auf Analysemöglichkeiten mit seinem Programm GROGRA ein.

2.3 Alternativen zu L-Systemen

L-Systeme stellen ein diskretes mathematisches Modell dar. Biologen nutzen alternativ zu L-Systemen auch ein kontinuierliches mathematisches Modell, um das Verhalten von Zellen zu modellieren, vergleiche dazu [Ab86].
In diesem Kapitel werden nur die Modelle vorgestellt, die auch auf Grammatiken basieren und eine gewisse Ähnlichkeit zu L-Systemen aufweisen, bzw. L-Systeme wesentlich erweitern.
Einen Kompromiss zwischen L-Systemen und den Sprachen der Chomskyhierarchie stellen die *Indian Parallel Systems* dar. Hierbei wird, wie bei den Sprachen der Chomskyhierarchie, eine Produktion ausgesucht. Diese Produktion wird dann auf alle Symbole angewandt, die zu der Produktion passen. [Kud86]

2.3.1 P-Systeme

P-Systeme basieren auf einem weiteren mathematischen Modell, das sich an die Natur anlehnt. Eingeführt wurden die P-Systeme von Gheorghe Paun in [Pa98]. Das biologische Vorbild ist die Membrane, statt von P-Systemen spricht man deshalb auch von „Membrane Computing“. Dabei sollen P-Systeme biologische Membranen nicht modellieren, sondern einige Eigenschaften dieser Membranen nutzen.

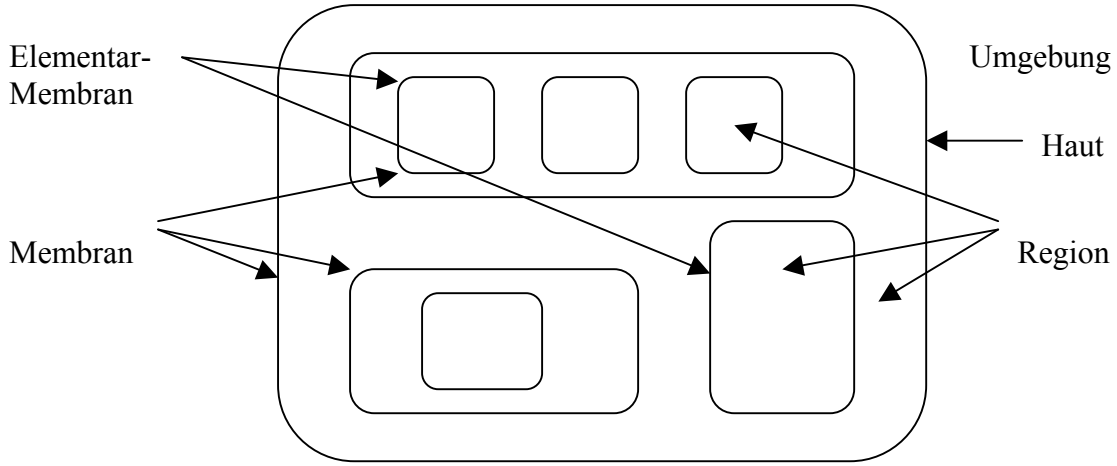


Abbildung 2.16: Beispiel einer Membran

Das System besteht aus einer Basismembran, deren Kanten die Haut darstellen. Auf diese Weise wird die Basismembran von ihrer Umgebung getrennt. Die Basismembran kann weitere Membranen enthalten. Somit erhält man eine hierarchische Struktur. Sind in der Region einer Membran keine weiteren Membranen enthalten, wird sie als Elementarmembran bezeichnet. Jede Membran hat eine Region, in der sich Objekte und Regeln befinden. Dabei stellen die Objekte Buchstaben eines Alphabetes dar und die Regeln die Produktionen. Die Anwendung der Produktion findet im Basismodel, wie in einem 0L-System, parallel und nicht deterministisch statt. .
Objekte können von einer Membran zu einer anderen weitergereicht werden. Membranen selbst können ihre Durchlässigkeit verändern, sich aufblähen und teilen.
Für weitere Informationen, empfiehlt es, sich die Homepage der P-Systeme zu besuchen, http://psystems.disco.unimib.it/ (Stand: 05/2004). [PR01] [Pa00]

2.3.2 Eco-Grammar Systeme

Eco-Grammar Systeme werden 1994 in *„Eco (grammar) systems“* von Păun, Kelemenová, Kelemen und Csuhaj-Varjú [Cs02] eingeführt. Die folgende Abbildung illustriert ein solches Eco-Grammar System.

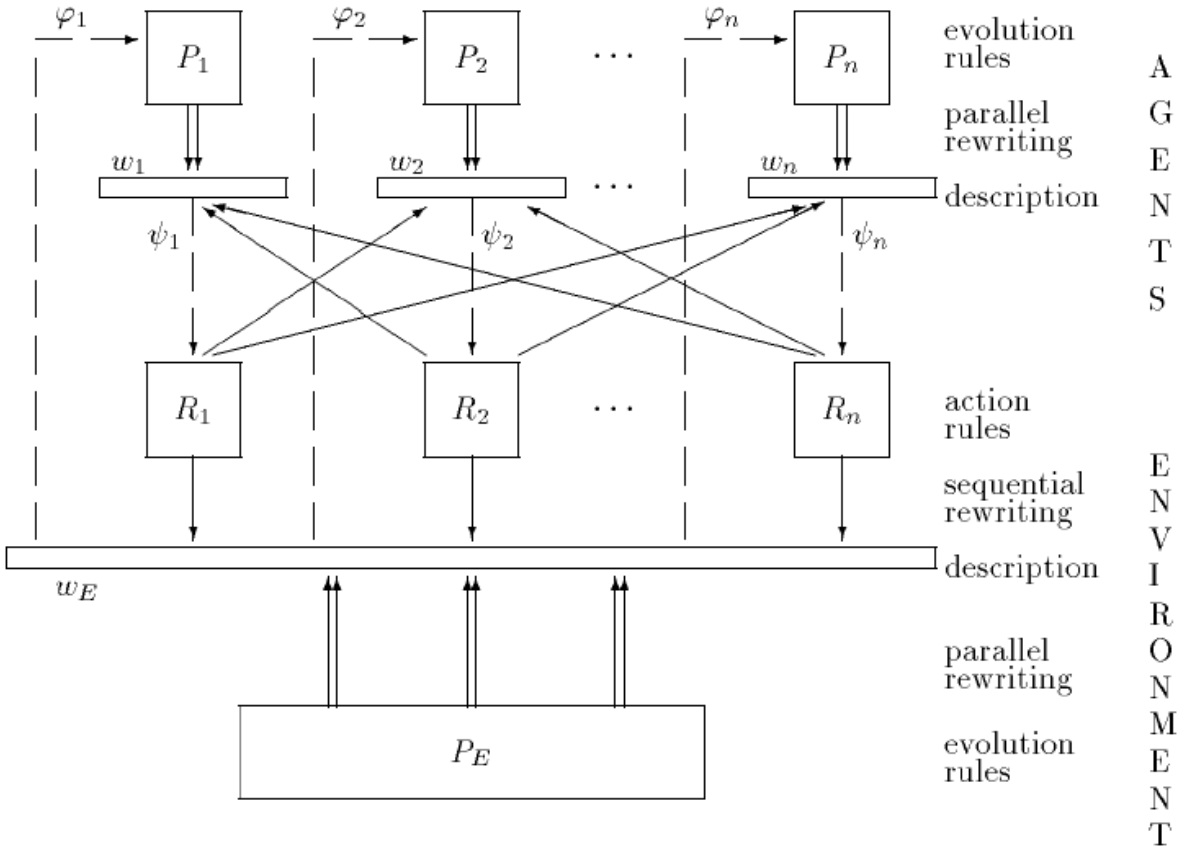


Abbildung 2.17: Ein vollständiges Eco-Grammar System [CPS95]

Die Idee, die hinter den Eco-Grammar Systems steckt, ist die Modellierung eines Ökosystems. In einem Eco-Grammar System gibt es zwei Rollen, die eines Agenten (Agents) und die Rolle der Umgebung (Environment). Dabei darf es eine beliebige Menge von Agenten geben, aber nur eine Umgebung. Alle Agenten leben in einer gemeinsamen Umgebung.
Die Umgebung kann sich selbst unabhängig von den Agenten weiterentwickeln, die Agenten dagegen entwickeln sich abhängig von der Umgebung. Dadurch nehmen die Agenten die Umgebung wahr. Die Umgebung selbst muss die Agenten nicht wahrnehmen, sie kann aber durch die Agenten verändert werden. Der Zustand eines jeden Agenten und der der Umgebung wird durch eine Zeichenkette repräsentiert. Die Agenten und die Umgebung stellen jeweils ein L-System dar, welches für sich selbst definiert ist. Außerdem existiert eine globale Zeit für das Ökosystem. Zu jedem Zeitschritt führen die Agenten und die Umgebung eine Ableitung ihres Zustandes aus. Für die Agenten ist immer nur eine Teilmenge der Produktionen „aktiv“, dies ist abhängig vom Zustand der Umgebung. Mit diesen Produktionen leiten die Agenten ihren neuen Zustand ab.
Im zweiten Schritt agieren die Agenten auf die Umgebung mit speziellen „action rules“. Dabei kann ein Agent nur eine Produktion in einem Zeitschritt auswählen. Die Auswahl ist dabei abhängig vom Zustand des Agenten. Die Aktion, die ein Agent mit der Produktion auf die Umgebung ausübt, hat Priorität gegenüber der Entwicklung der Umgebung. Die Symbole der Umgebung, die durch diese Aktion nicht berührt wurden, entwickeln sich weiter, indem sie eine Produktion aussuchen und diese anwenden. Die „action rules“ wirken sich auch auf die Zustände der anderen Agenten aus. [CPS95] [Cs02]

2.4 Genetische Algorithmen

Im Folgenden wird eine kurze Übersicht zum Thema Genetische Algorithmen gegeben. Sie stellen die Basis dar, auf der später die Mutation von L-Systemen besprochen werden. Ausführliche Literatur zu dem Thema ist in [Mi99] [Ja97] zu finden.
1975 führte John Holland in „Adaption in Natural and Artificial Systems“ die Genetischen Algorithmen ein.
Die genetischen Algorithmen kommen aus der Genetik. Dabei enthält ein Chromosom, auch Genotyp genannt, die Erbinformationen. Das Individuum, das entsteht, wird Phänotyp genannt.
Genetische Algorithmen werden benutzt, um für ein Problem, für das auf üblichen Wegen keine Lösung gefunden wird, in einer akzeptablen Zeit eventuell doch noch eine Lösung zu finden.
In GA werden die Chromosomen durch Bit-Strings repräsentiert, wobei jedes Bit den Wert 0 oder 1 einnehmen kann. In einer Programmiersprache wie Pascal würde man ein boolesches Array zur Repräsentation eines Chromosoms nehmen.
Der prinzipielle Ablauf eines GA ist in Abbildung 2.17 illustriert.

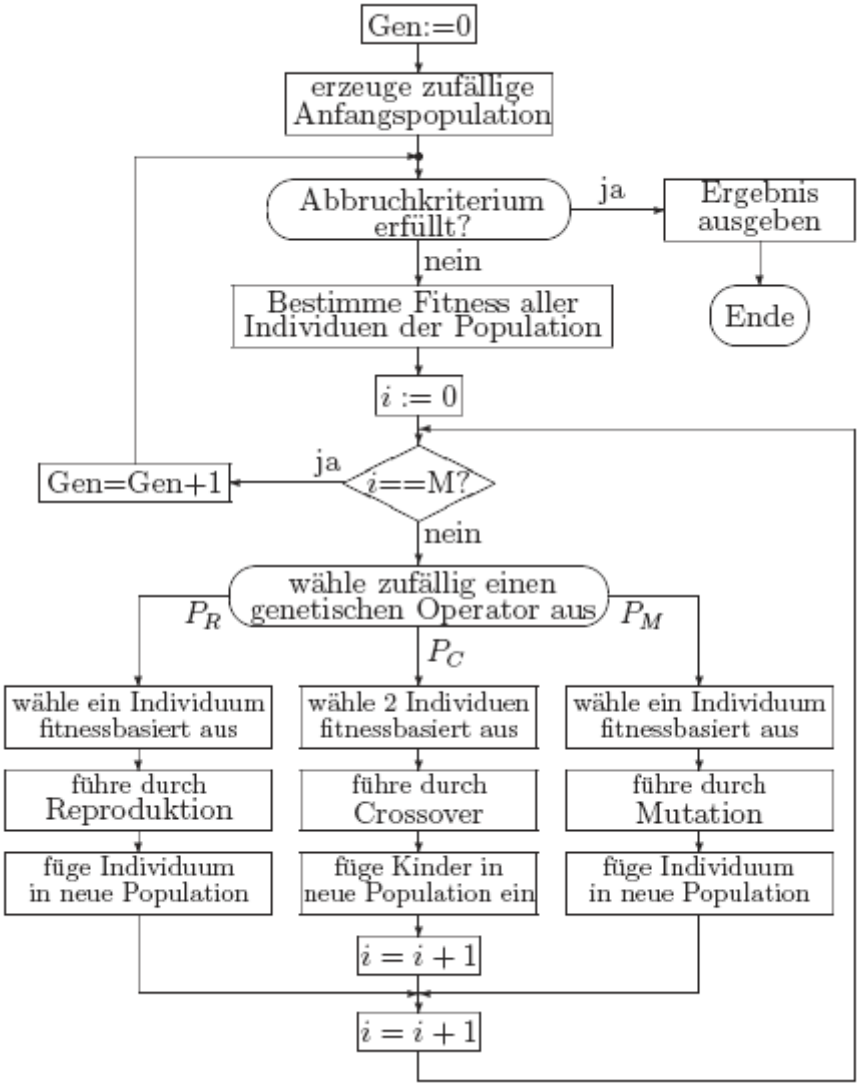


Abbildung 2.18: PAP eines GA [Sch03]

Am Anfang wird eine zufällige Anfangspopulation erzeugt, das heißt eine nicht leere Menge von Chromosomen, und geprüft, ob diese schon das Problem löst. Sollte dies nicht der Fall sein, dann wird die Fitness eines jeden Individuums der Population bestimmt. Dabei wird jedem Individuum ein Maß zugeschrieben, das die Nähe zur Lösung beschreibt.
Per Zufall wird eine Operation ausgewählt. Entsprechend der Operation werden zwei Individuen oder nur ein Individuum benötigt. Um nun diese Individuen aus der Population herauszusuchen, wird eine Selektion durchgeführt. Bei dieser Selektion sind mehrere Varianten möglich.

- *Tournament:*
  Eine bestimmte Anzahl von willkürlich gewählten Individuen wird zu einer Gruppe zusammengestellt. Auf der Basis der Fitness der Individuen in der Gruppe werden, entsprechend der Operation, ein oder zwei Individuen selektiert.
- *Roulette:*
  Die Auswahlwahrscheinlichkeit steigt proportional zur Fitness.
- *Overselection:*
  Die Auswahlwahrscheinlichkeit wird so manipuliert, dass das stärkste Drittel der Population eine Gewichtung von 80 % und der Rest eine Gewichtung von 20 % bekommt.

Nun wird der Operator auf die Individuen angewandt. Bei den GA existieren drei verschiedene Operatoren, die sich an die Natur anlehnen.
Der erste GA-Operator ist die Reproduktion. Bei der Reproduktion wird das Individuum nicht verändert und in die neue Population eingefügt.
Beim Operator Crossover werden zwei Individuen verändert und in die neue Population eingefügt. Dabei wird ein Teil des Bit-Strings zwischen den beiden Individuen vertauscht. Beide Bit-Strings müssen aber die gleiche Länge aufweisen.

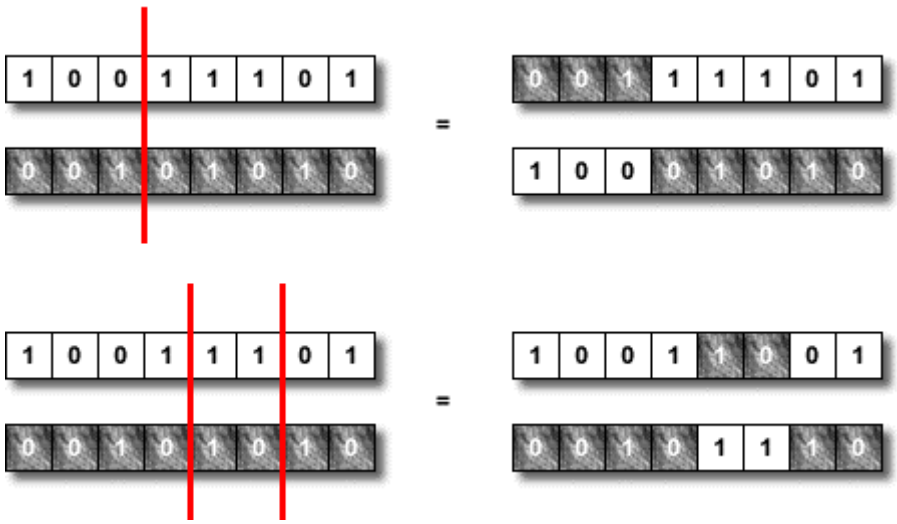


Abbildung 2.19: Crossover in GA

Der Crossover Operator kann so modifiziert werden, dass mehr als zwei Individuen benutzt werden, dass mehrere Teile des Bit-Strings der Individuen vertauscht werden oder dass beides gleichzeitig stattfindet.
Der letzte Operator ist die Mutation. Bei der Mutation werden an zufälligen Stellen des Bit-Strings die Bits gekippt. Die Anzahl der zu kippenden Bits muss vorher festgelegt werden. Das mutierte Individuum wird dann in die neue Population eingefügt.

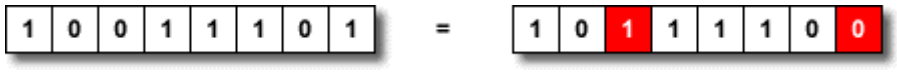


Abbildung 2.20: Mutation in GA

Der Genetische Algorithmus bricht idealerweise bei einer Lösung ab. Es sollte dem Anwender trotzdem jederzeit die Möglichkeit zur Verfügung stehen, den GA abzubrechen und auf die zu diesem Zeitpunkt beste Lösung zurückzugreifen, denn die Laufzeit des GA kann sehr lang sein. [Bo04] [Ja97] [Mi99] [Sch03] [Ban03]

2.5 Genetische Programmierung

In diesem Kapitel wird nur so weit ein Überblick über die Genetische Programmierung gegeben, wie es für das Verständnis der Mutation von L-Systemen notwendig ist. Eine ausführlichere Erörterung dieses Themas findet sich bei [BNKF98].
Die früheste Form der Genetischen Programmierung geht auf Smith (1980) zurück. Er arbeitete mit Bit-Strings, wie es bei den Genetischen Algorithmen der Fall ist, mit dem Unterschied, dass die von ihm angewendeten Bit-Strings eine variable Länge hatten. In den Folgejahren werden weitere frühe Formen der Genetischen Programmierung entwickelt, bis Koza 1992 die jetzige Form der Genetischen Programmierung einführte.
Wie die GA wird auch die Genetische Programmierung benutzt, um ein Problem zu lösen, wenn dies mit herkömmlichen Lösungsmethoden nicht oder nicht in einer akzeptablen Zeit erreicht werden kann. Beispiele für den Anwendungsbereich sind Intrusion Detection, Evolution von Hardware-Eigenschaften und Klassifikation von Bildern in Geo-Systemen.
Bei der Genetischen Programmierung werden Programme einer Evolution unterzogen. Die Population ist eine Menge von Programmen. Da sich ein Programm nicht in Bit-Strings formulieren lässt, müssen dazu andere Datenstrukturen eingesetzt werden.

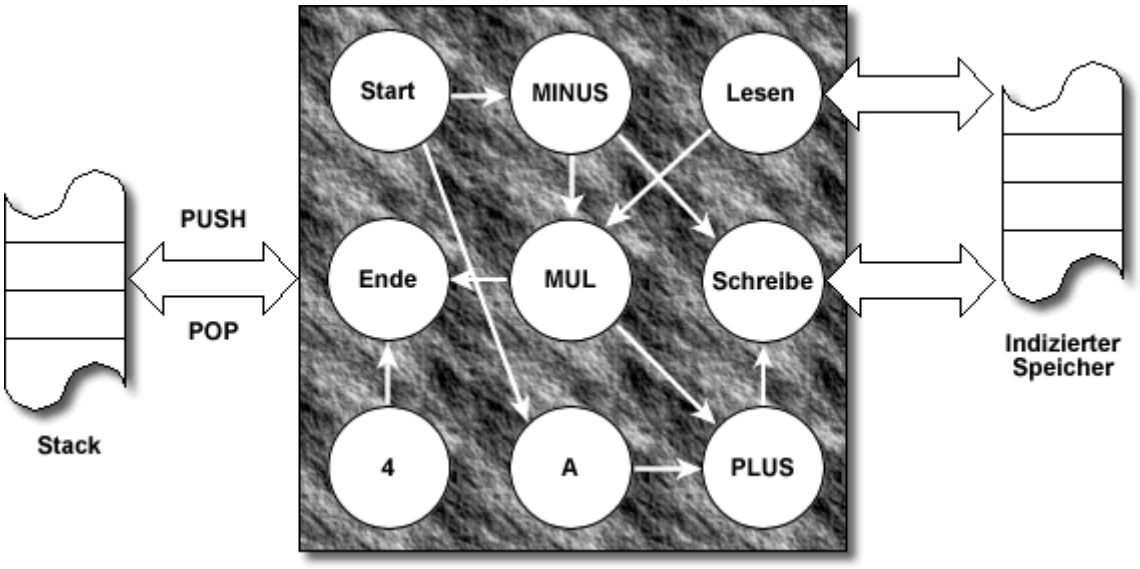


Abbildung 2.21: Automat als Datenstruktur

Die drei gängigsten Datenstrukturen sind der Baum, eine lineare Sequenz von Befehlen und ein Automat. Beim Einsatz von Automaten kann man sich drei Bereiche vorstellen, eine CPU, einen Stackspeicher und einen indizierten Speicher. Dabei ist die CPU der eigentliche Automat, in dem ein Start- und ein Endzustand existiert sowie Zustände, die eine arithmetische oder logische Funktion ausführen. Ein Zustand kann auch ein Unterprogramm darstellen. Durch die Verbindung von Zuständen mit Kanten wird ein Programm erzeugt. Ein GP-System auf der Basis von Automaten ist PADO. Leider existiert zu PADO keine Web-Präsenz, daher werden alternativ die Links zu Astro Teller (http://www-2.cs.cmu.edu/~astro/astro-papers.html (Stand: 06/2004)) und Manuela M. Veloso (http://www-2.cs.cmu.edu/~mmv/ (Stand: 06/2004)) angeboten, beide sind die Entwickler von PADO.

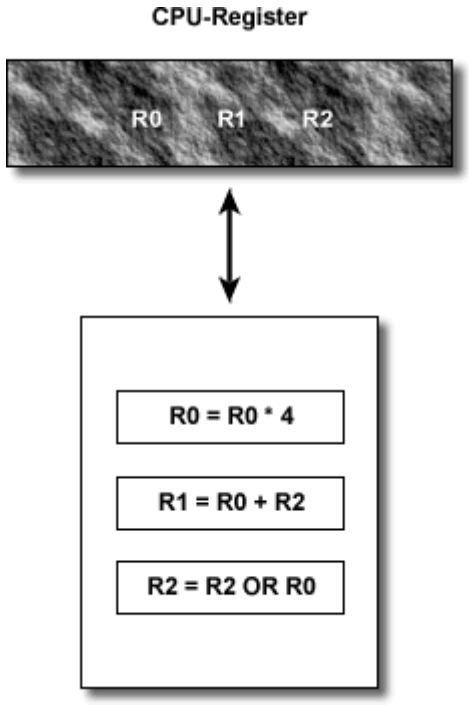


Abbildung 2.22: Lineare Struktur

Bei der linearen Sequenz von Befehlen wird ein Computer nachgebaut. So besteht ein Programm aus einer CPU mit einer bestimmten Anzahl von Registern, die eine lineare Sequenz von Befehlen abarbeitet. In den Befehlen sind nur arithmetische und logische Operationen möglich. AIMGP war solch ein GP-System. Aus AIMGP ist Discipulus entstanden. Weitere Informationen sind unter http://www.aimlearning.com (Stand: 06/2004) zu finden.

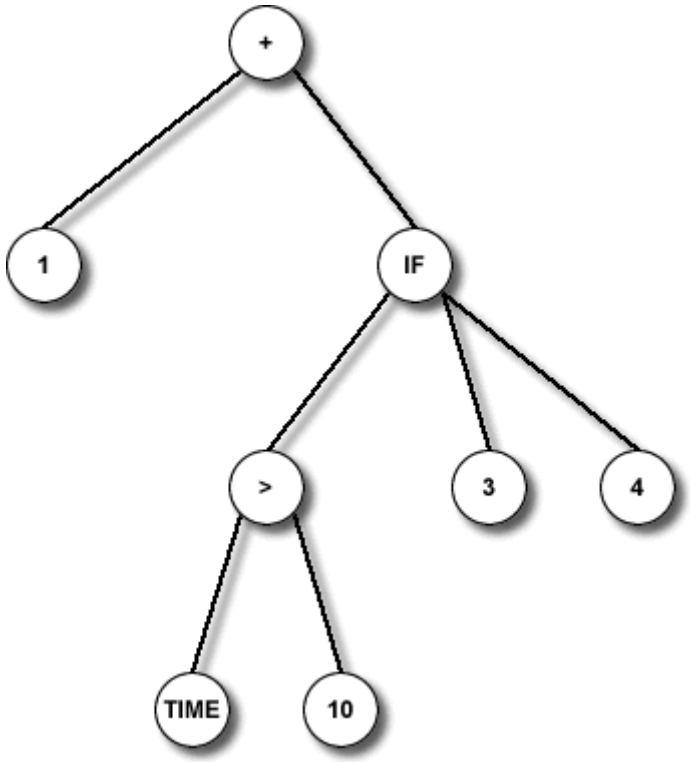


Abbildung 2.23: Bäume als Datenstruktur

Bäume als Datenstruktur für Programme sind die gebräuchlichste Lösung. Dabei stellen die Blätter Variablen oder Konstanten dar und die restlichen Knoten arithmetische oder logische Operationen. Der Vorteil von Bäumen ist, dass auch ein IF THEN/ELSE Konstrukt verwendet werden kann. Die Bäume sind dabei die Repräsentation eines Ausdruckes in Präfix-Notation. So steht der Baum in Abbildung 2.22 für den Ausdruck

```
( + 1 ( IF ( > TIME 10 ) 3 4 ) )
```

in Präfix-Notation. Der Ausdruck in imperativer Form ist

```
if (TIME > 10) then
     return 1 + 3;
else
     return 1+4;
```

Diese Bäume nennt man auch S-Expression. Die Kombination aus S-Expression und GA ergibt dann GP. Daher ist der prinzipielle Programmablauf identisch mit dem der GA, siehe Abbildung 2.17. Der Unterschied liegt in den Operatoren und in deren Ausführung auf die Datenstruktur.
Die Reproduktion eines Individuums ist einfach. Eine Kopie des Individuums wird in die nächste Population eingefügt.
Auch der Crossover-Operator bei der linearen Sequenz von Befehlen ist unproblematisch. Da alle Programme auf der gleichen CPU laufen, können Sequenzen einfach vertauscht werden. Bei Bäumen wird ein Teilbaum von zwei Individuen selektiert und vertauscht.

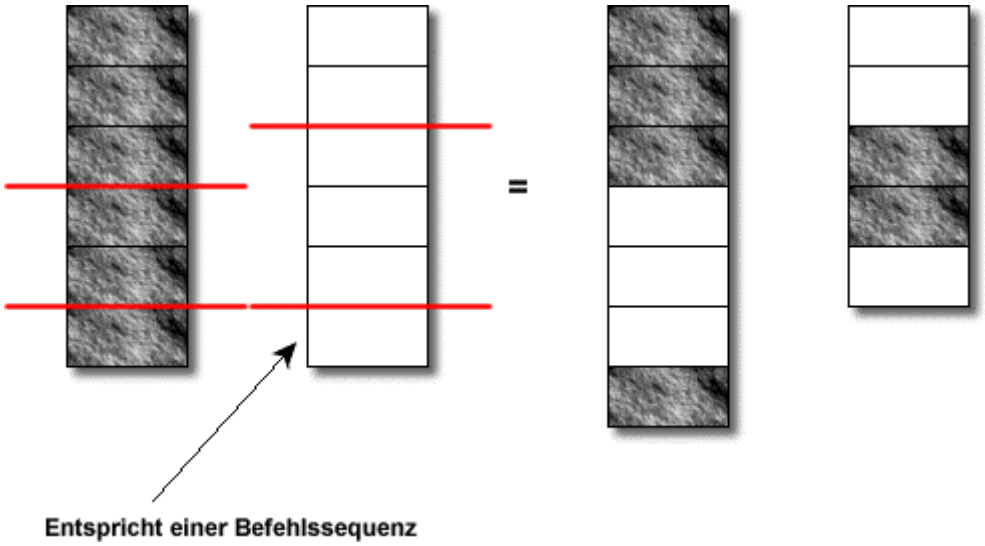


Abbildung 2.24: Crossover in GP – Lineare Befehlssequenz

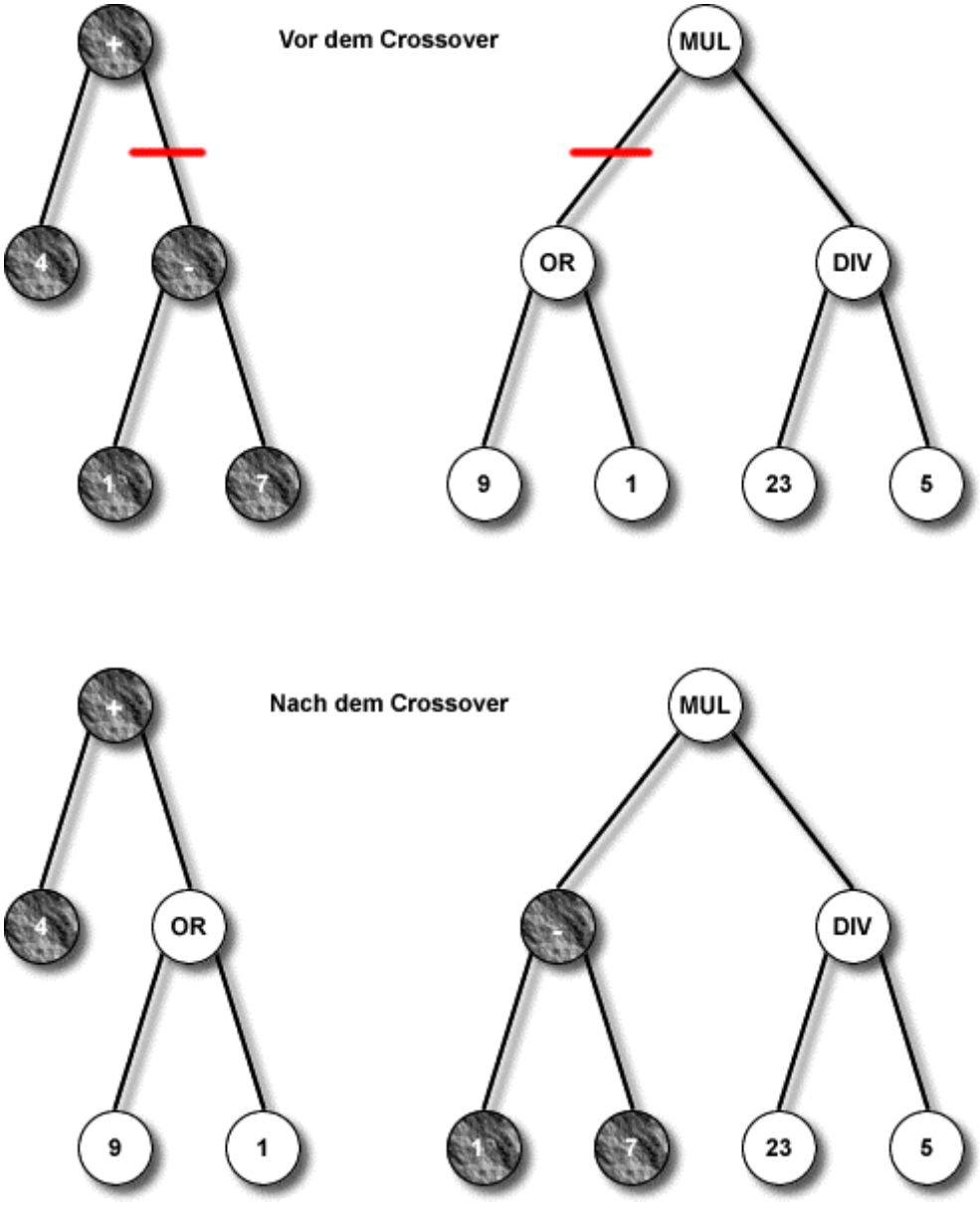


Abbildung 2.25: Crossover in GP - Baum

Problematischer wird es bei der Mutation von Bäumen. Hier gibt es zwei Möglichkeiten der Mutation. Bei der *Tree Mutation* wird ein bestehender Teilbaum durch einen neuen zufälligen Teilbaum ersetzt. Die zweite Möglichkeit ist die *Linear Mutation*. Dabei wird ein Knoten des Baumes ausgewählt und durch einen neuen zufälligen Knoten ersetzt, wobei der neue Knoten sich in die Präfix-Notation eingliedern lassen muss. So kann ein einzelner Operand nicht durch eine einzelne Operation ersetzt werden und umgekehrt.

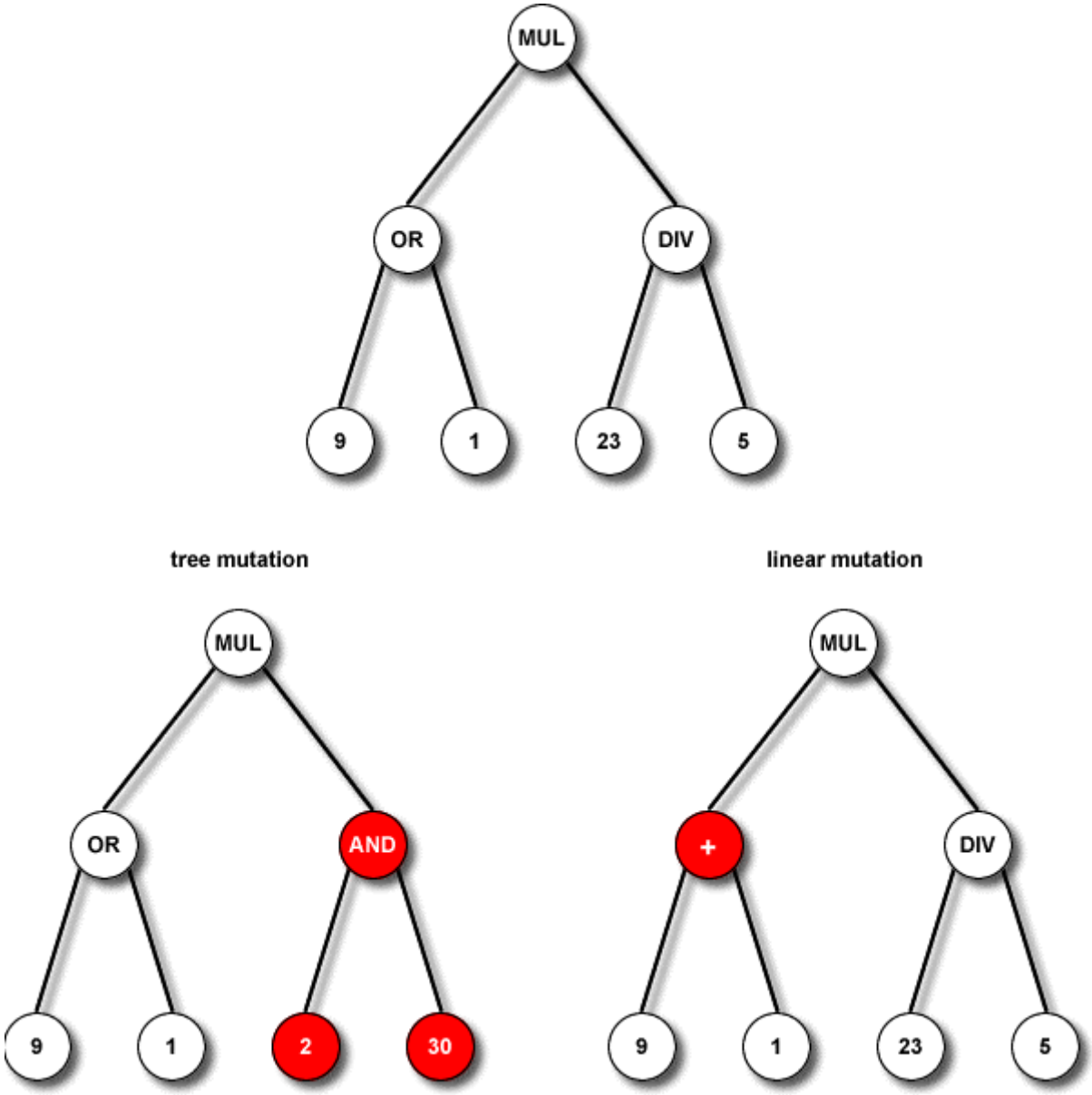


Abbildung 2.26: Mutation beim Baum

Bei der Mutation von linearen Befehlssequenzen wird zu Anfang eine komplette Sequenz herausgesucht. Im Anschluss wird per Zufall eine der drei folgenden möglichen Mutation gewählt.

1. Ein Register wird durch ein anderes existierendes Register ersetzt.
2. Eine Konstante wird durch eine andere im Wertebereich liegende Konstante geändert.
3. Die Operation wird durch eine andere erlaubte Operation ersetzt.

Beispiel 2.17:
Folgende Sequenz soll mutieren, wobei die CPU die Register R0, R1 und R2 besitzt und die arithmetischen Operationen +, -, MUL und DIV versteht sowie die bit-orientierten logischen Operationen AND, OR und NOT. Der Wertebereich für die Konstanten liegt im Wertebereich der ganzen nicht negativen Zahlen.

```
R0 = R1 + 4
```

Mögliche Mutationen:

```
R1 = R1 + 4              (1. Fall)
R0 = R2 + 4              (1. Fall)
R0 = R1 + 8              (2. Fall)
R0 = R1 MUL 4            (3. Fall)
```

Beim Umgang mit den Populationen gibt es zwei Ansätze. Entweder wird bei jedem Durchlauf des Algorithmus durch Selektion und die Anwendung eines Operators eine komplett neue Population erzeugt oder das neue Individuum wird in die bestehende Population eingegliedert, wofür ein anderes Individuum weichen muss. Den zweiten Ansatz nennt man auch *steady-state*. Er hat den Vorteil, dass zu jedem Zeitpunkt die bisher beste Lösung verfügbar ist. Bei der Erzeugung einer neuen Population kann die bisher beste Lösung verloren gehen. Diese Ansätze gelten auch für die GA. Wie die GA ist auch die GP sehr rechenaufwendig. [BNKF98] [Ban03] [Bo04] [Ja97]

2.6 Mutation von L-Systemen

Hier sollen einige Gedanken zu der Frage aufgegriffen werden, wie L-Systeme mutieren können. Dabei werden zwei Ansätze betrachtet, der Einsatz von GA und der Einsatz von GP. Gerade L-Systeme, die eine Erweiterung zum 0L-System darstellen, sind sehr interessant für die Mutation. So könnte bei einem T0L-System durch Mutation die Umgebung manipuliert werden. Bei stochastischen L-Systemen könnte die Wahrscheinlichkeitsverteilung auf die Regeln variiert werden. Bei IL-Systemen könnte der Kontext verändert werden. In der Praxis werden zur Mutation am häufigsten parametrisierte PDB0L-Systeme verwendet, weil sich hiermit komplexere Objekte realisieren lassen. Im praktischen Einsatz ist es empfehlenswert, ein deterministisches L-System oder ein stochastisches zu verwenden, damit der Anwender nicht in jeden Ableitungsschritt eingreifen muss.

2.6.1 Mutation von L-Systemen unter Genetischen Algorithmen

Bei der Anwendung von GA mit L-Systemen muss erst bestimmt werden, was ein Chromosom ist. Entgegen der eigentlichen Definition eines Chromosoms, dass es ein Bit-String mit einer fixen Länge ist, stellt eine Produktion ein Chromosom dar. Dieses Chromosom hat keine feste Länge. Viele Autoren (zum Beispiel Fox [Fo00] und Mock [Mo96]) sprechen trotzdem von GA, nur wenige sprechen von Evolutionären Algorithmen (zum Beispiel Kókai [KTV99a] und Hornby [Ho03]). Da die Idee auf GA basiert, wird auch hier von GA gesprochen.

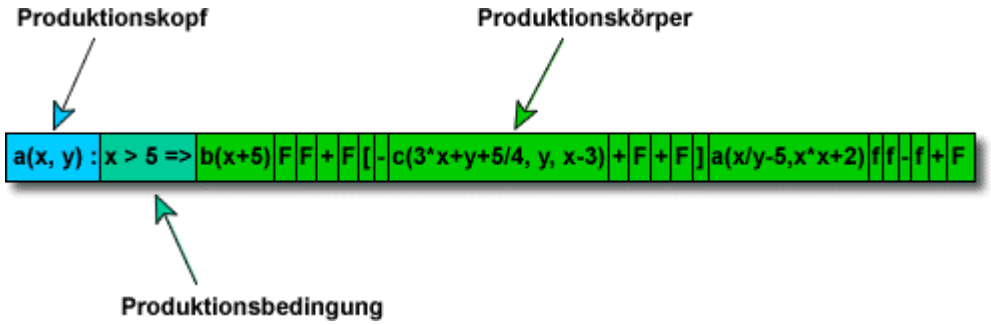


Abbildung 2.27: Eine Produktion als Chromosom

Die Abbildung von Evolution auf L-Systeme unter der Nutzung von GA hat sich als die gängigste Variante herausgestellt. Dabei werden am häufigsten parametrisierte PDB0L-Systeme verwendet. Daher werden die folgenden Mutationsmöglichkeiten auch an solchen Produktionen gezeigt. Die Erweiterung der Mutation auf andere L-Systeme sollte dann nicht mehr schwer fallen.

Point Mutation:
Hierbei wird nur ein einzelnes Symbol oder eine Zahl eines L-Systems angefasst. Es können auch mehrere einzelne Symbole oder Zahlen verändert werden, sie stehen bei dieser Mutationsart nur nicht in einem bestimmten Kontext.
Bezogen auf die Argumente, die in parametrisierten L-Systemen übergeben werden, heißt es, das Zahlen verändert werden können oder eine Variable durch eine andere ersetzt werden kann. Es sei daran erinnert, dass die Variablen in einer Produktion benutzt werden müssen, die als Formalparameter angegeben wurden, ansonsten wäre die Produktion ungültig. Neben dem Ersetzen können auch Argumententeile gelöscht werden. Dabei muss der Operator mitgelöscht werden. In den Argumenten können auch neue Zahlen oder vorhandenen Variablen mit einem Operator neu hinzugefügt werden. Das Verändern der Anzahl der Argumente ist eine problematische Aufgabe. Denn dadurch müssen alle anderen Produktionen, die auf diese Produktion zugreifen, entsprechend auch verändert werden. Auch ein gültiger Ausdruck für die Formalparameter muss der veränderten Produktion übergeben werden.
Das Einfügen, Löschen und Ersetzen kann man auch auf einzelne Symbole der Produktion beziehen. Dabei ist zu beachten, dass beim Löschen auch der Argumententeil mitgelöscht wird. Beim Ersetzen eines Symbols durch ein anderes muss gesichert werden, dass die Argumente auch zur Produktion des Symbols passen. Sollte ein Symbol in eine Produktion eingefügt werden, dann muss auch ein entsprechender Argumententeil generiert werden. Das Ersetzen eines Symbols auf der linken Seite einer Produktion würde in einem deterministischen System bedeuten, dass zwei Symbole vertauscht werden und dass die jeweilige rechte Seite geprüft werden muss, wenn die Anzahl der Argumente beider Symbole ungleich ist. Dementsprechend müssten auch alle anderen Produktionen, die das Symbol enthalten, verändert werden.
Insgesamt ist es nicht ratsam, die Anzahl von Argumenten oder das Symbol auf der linken Seite einer Produktion zu mutieren. Es ist auch nicht ratsam, eckigen Klammern einer Point Mutation zu unterziehen, denn ein Wort ist nur interpretierbar, wenn die Anzahl der offenen und geschlossenen Klammern gleich ist. Zu überprüfen, ob das Verändern einer Klammer Auswirkungen hat, ist nur sehr schwer möglich.

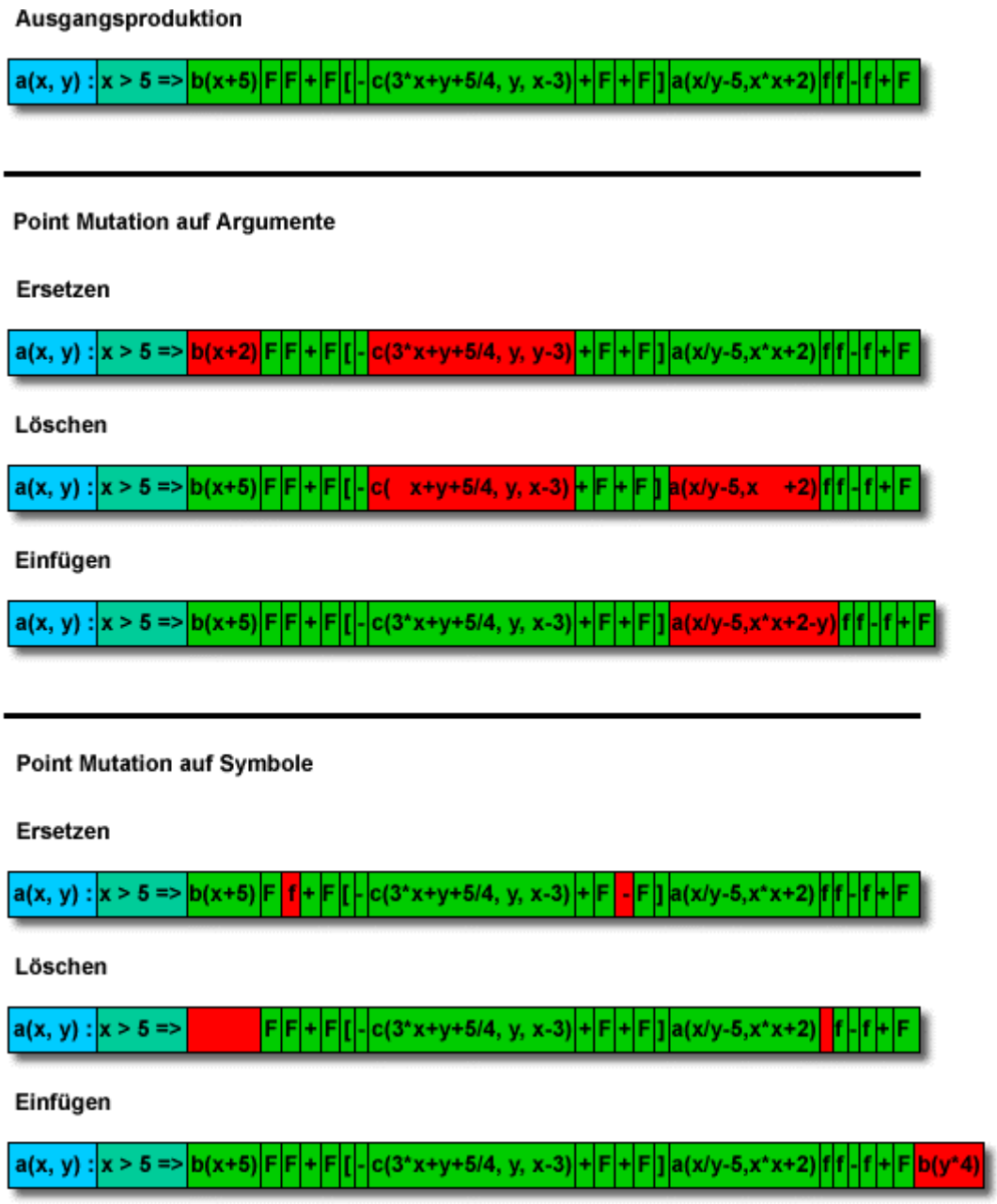


Abbildung 2.28: Beispiele für Point Mutation

Die folgenden Mutationsarten beziehen sich immer auf die linke Seite der Produktion, mit Ausnahme der letzten.

Inversion:
Hierbei wird ein zusammenhängender Teil der Produktion herausgenommen, die Reihenfolge der Symbole umgekehrt und wieder an die gleiche Stelle eingefügt. Das Einbeziehen von eckigen Klammern sollte dabei verhindert werden. Dadurch ist die Wahrscheinlichkeit sehr hoch, dass ein ungültiges Wort generiert wird. Argumente können innerhalb ihres Argumententeils invertiert werden, es ist dabei nur zu beachten, dass alle Operanden mit einem Operator gültig verknüpft sind und dass ein gültiger Ausdruck entsteht.

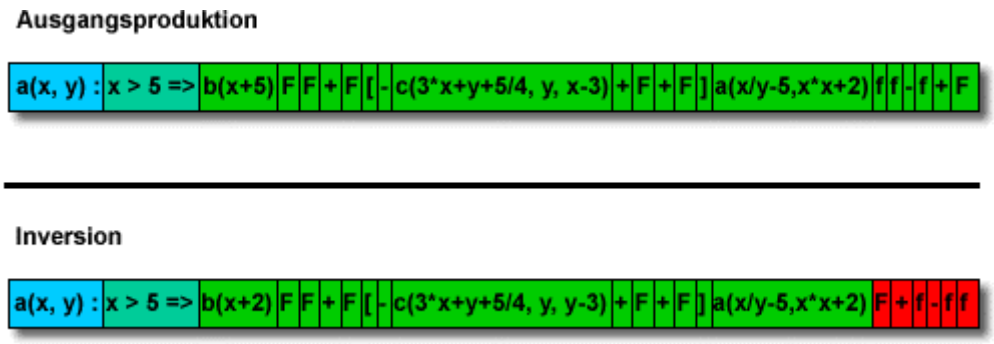


Abbildung 2.29: Inversion einer Produktion

Deletion:
Bei der Deletion wird ein zusammenhängender Teil der Produktion aus mehreren Symbolen entfernt. Eckige Klammern können mitgelöscht werden, wenn es paarweise geschieht, um sicherzustellen, dass das generierte Wort gültig ist. Beim Anwenden von Deletion auf einen Argumententeil eines Symbols ist darauf zu achten, dass der Argumententeil gültig bleibt.

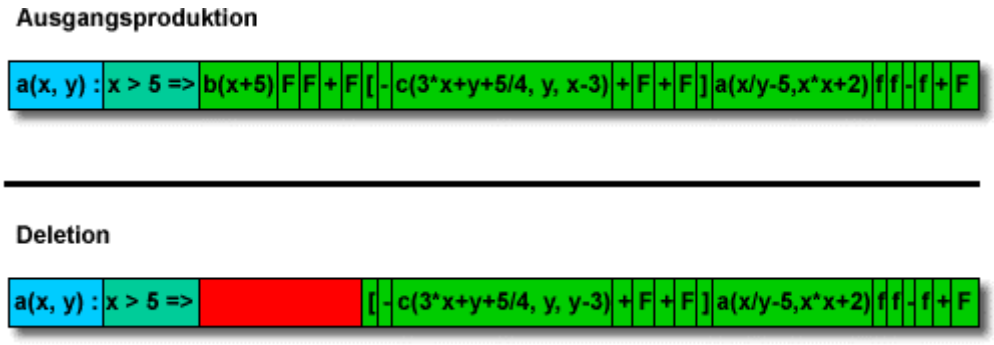


Abbildung 2.30: Deletion bei einer Produktion

Duplication:
Ein zusammenhängender Teil der Produktion wird dupliziert und an einer zufälligen Stelle der Produktion eingefügt. Auch hier ist nur das paarweise Duplizieren von eckigen Klammern zulässig, um die Gültigkeit des Wortes zu garantieren. Das Duplizieren eines Argumententeils und das Einfügen in einem Argumententeil ist nicht ratsam, weil dadurch die komplette Produktion verändert werden müsste.

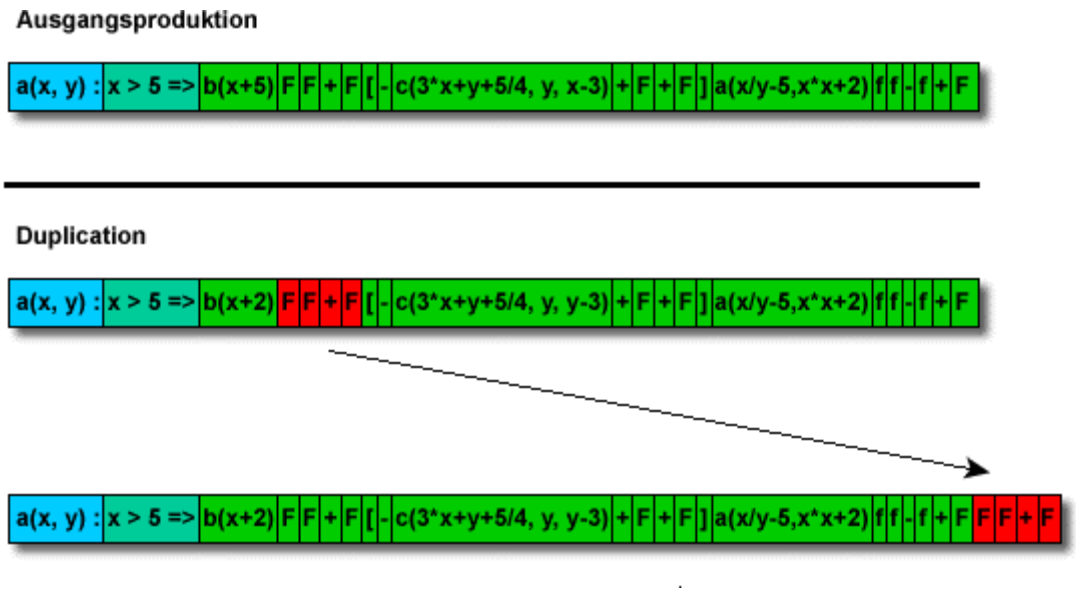


Abbildung 2.31: Beispiel für Duplication an einer Produktion

Translocation:
Hierbei wird ein zusammenhängender Teil der Produktion herausgelöst und an einer zufälligen Stelle in der Produktion eingefügt. Auch hier ist nur das paarweise Verschieben von eckigen Klammern erlaubt, um die Gültigkeit des Wortes zu garantieren. Eine Translocation von einem Argumententeil ist nicht möglich, weil dann ein neuer Argumententeil für ein Symbol generiert werden müsste. Das komplette Vertauschen zweier Argumententeile wäre schon eher denkbar wenn gesichert wird, dass die Argumententeile zu den Symbolen passen. Denkbar wäre auch das Vertauschen von Teilen der Produktion.

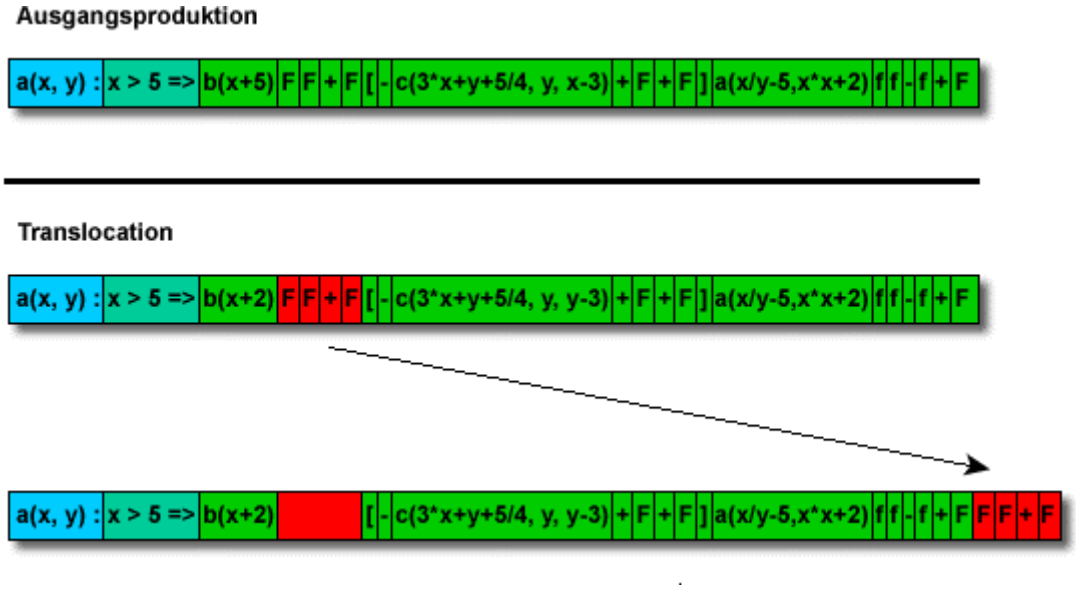


Abbildung 2.32: Mutation einer Produktion mit Translocation

Hinzufügen eines neuen Symbols:
In einer Produktion wird ein Symbol hinzugefügt, das bisher noch nicht in diesem L-System existiert. Das hat zur Folge, dass eine entsprechende Produktion generiert werden muss.

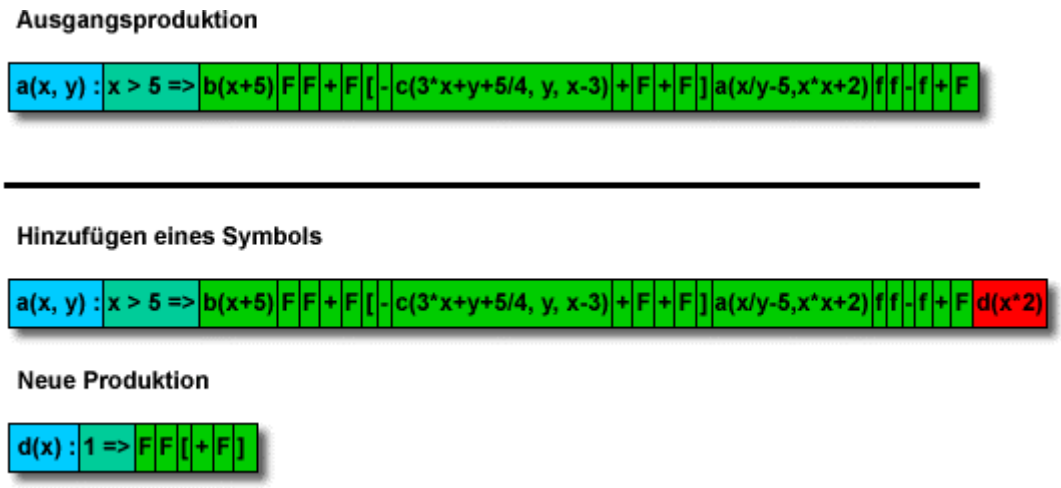


Abbildung 2.33: Hinzufügen eines neuen Symbols

Mutation der Bedingung:
Es ist auch denkbar, dass die Bedingung einer Produktion mutiert werden kann. Das Problem hierbei ist zu gewährleisten, dass zu jedem Zeitpunkt eine Produktion für ein Symbol greift. Ansonsten wäre es denkbar, jegliche Bedingung mutieren zu lassen.

Das Problem der eckigen Klammern könnte bei der Mutation auch vernachlässigt werden, wenn vor der graphischen Interpretation das Wort geprüft wird und entsprechende eckige Klammern nachträglich an sinnvollen Stellen hinzugefügt werden.
Wie deutlich wird, unterscheiden sich die Arten der Mutation anhand ihrer Auswirkungen auf das Chromosom. So zählt die *Point Mutation* zur Gruppe der Genmutationen. Ein Gen ist in diesem Fall ein Symbol mit seiner Parameterliste in der Produktion. Eine Ausnahme bildet der Bedingungsteil einer Produktion. Dieser wird als Einheit betrachtet, auch wenn die Bedingung noch so komplex ist. Daher würde eine Mutation auf die Bedingung auch zur Genmutation zählen.
Eine weitere Gruppe bildet die Chromosomenmutation. Zu dieser Gruppe gehören alle Mutationsarten, die sich auf mehrere Gene auswirken. Damit zählen die Inversion, Deletion, Duplication und die Translocation auch zur Gruppe der Chromosomenmutation.
Die letzte Gruppe wird als Genommutationen bezeichnet. In diesem Fall bildet das komplette L-System das Genom. Eine Mutation gehört in diese Gruppe, wenn die Anwendung des Mutationsoperators die Anzahl der Chromosomen verändert. Dazu zählt auch der Operator *Hinzufügen eines neuen Symbols*, der die Anzahl der Chromosomen erhöht. Denkbar als Mutationsoperator wäre auch das Löschen eines Chromosoms. Dies würde eine Produktion löschen und alle Symbole aus den noch vorhandenen Produktionen entfernen, die auf die zu löschenden Produktionen verweisen. Allgemein besteht ein komplexeres L-System aus polyploiden Chromosomen, das heißt die Chromosomenanzahl ist größer 2 (Produktionen). Genome mit einem Chromosom werden als haploide Chromosomen und Genome mit zwei Chromosomen als diploide Chromosomen bezeichnet.
Die Auswirkung von Mutationen wird am Beispiel der Software *L-System* gezeigt, da es ein Modul zur Mutation enthält. Die Software selbst wird später noch genauer in Kapitel 3.2 besprochen, zu Demonstrationszwecken wird an dieser Stelle nur kurz darauf eingegangen.
[[LS4]]

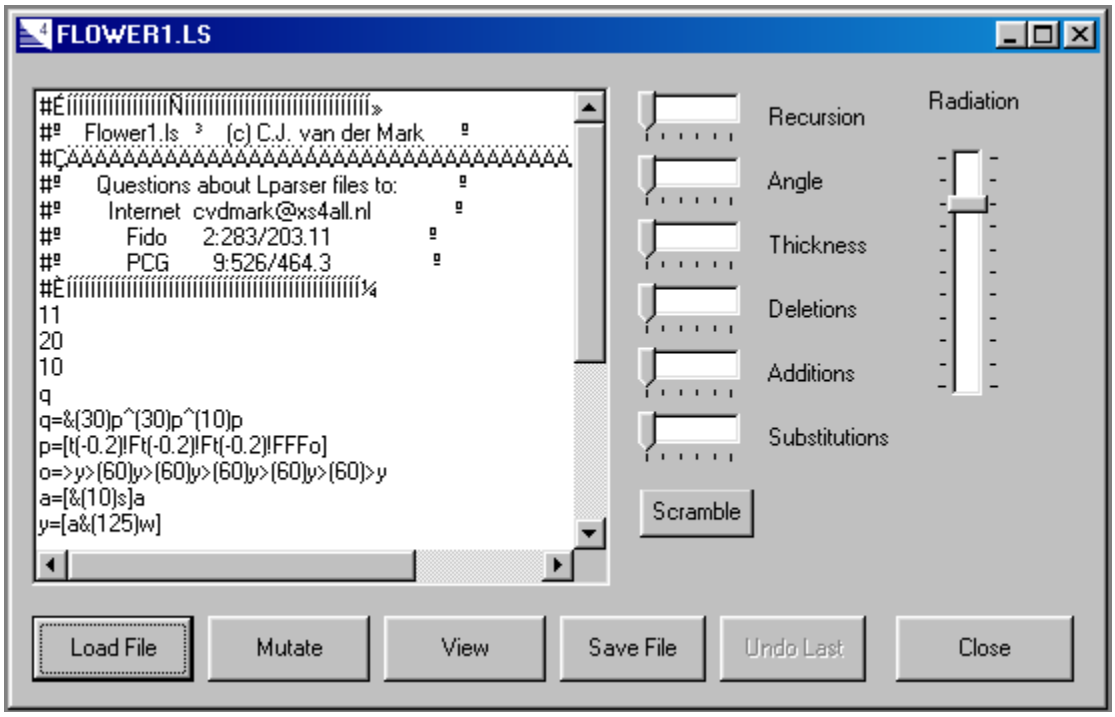


Abbildung 2.34: Der Mutator von L-System

Der *Mutator* beim *L-System* bietet die Möglichkeit, L-System zu mutieren und für die Mutation diverse Parameter festzulegen. Dabei werden folgende Mutationsarten unterstützt.

- *Recursion*
  Veränderung der Rekursionstiefe.
- *Angle*
  Mutation des Basiswinkels.
- *Thickness*
  Die Startdicke für die 3D-Grafik.
- *Deletions*
  Das Löschen von Teilen der Produktion.
- *Additions*
  Das Hinzufügen von Teilen in die Produktion.
- *Substitutions*
  Ersetzen von Teilen der Produktion.

Die Stärke jeder Mutationsart lässt sich gesondert einstellen. Außerdem kann über *Radiation* die globale Stärke der Mutation geregelt werden. Ansonsten gibt es einen Zufallsgenerator (Scramble), der die Verteilung der Stärken vornimmt.
Im Folgenden werden die Möglichkeiten der Mutation an einigen Beispielen verdeutlicht. Das ursprüngliche L-System heißt FLOWER1.LS und die normal erzeugte 3D-Grafik ist in Abbildung 2.34 zu sehen.

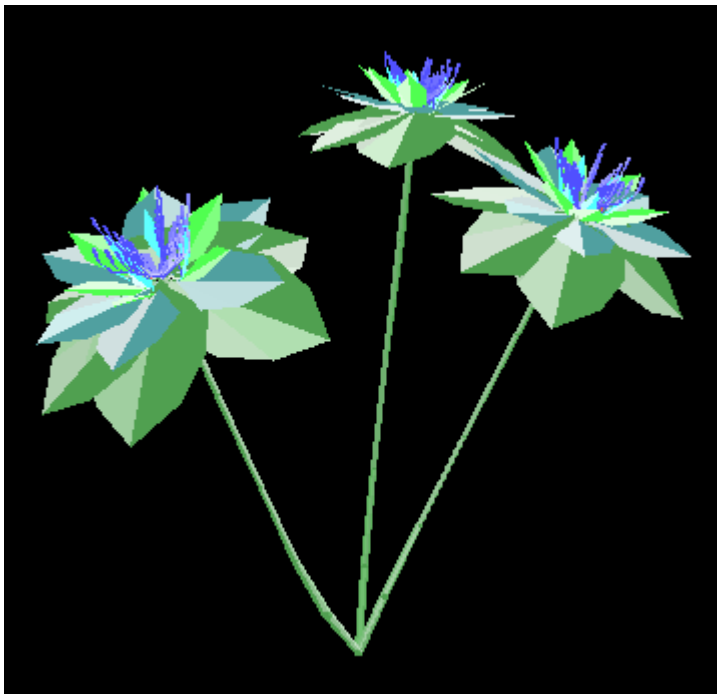


Abbildung 2.35: Die nicht mutierte Pflanze

Die Blume wurde zwei Mutationsschritten unterzogen. Dabei wurden nur zwei Mutationsarten aktiviert, Additions, volle Stärke, und Substitutions, mittlere Stärke.

Abbildung 2.36: Eine Mutation der Blume

Erneut wurden zwei Mutationsschritte durchgeführ, nur dieses Mal wurden Angle, Deletions und Substitutions sehr hoch angesetzt und Additions sehr niedrig.

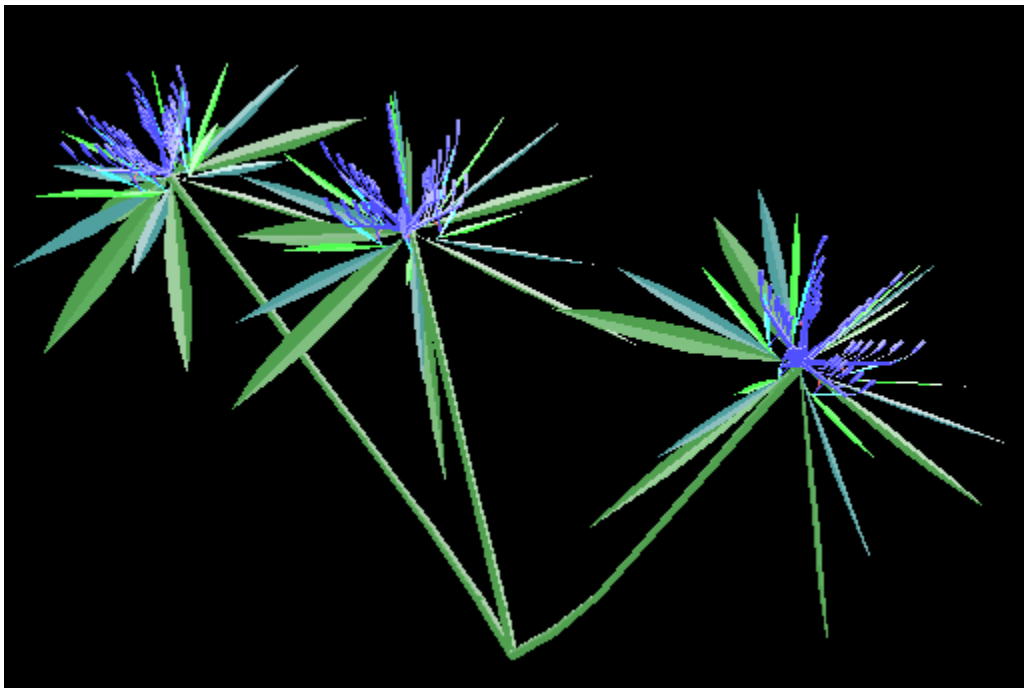


Abbildung 2.37: Eine zweite Mutation der Blume

Ein weiterer interessanter Aspekt wäre es, die Mutation dynamisch zu erstellen. Der übliche Ansatz ist eine statische Mutation, das heißt ein L-System erlebt vor Generierung des Wortes eine Mutation. Alternativ kann man auch während der Wortgenerierung nach drei Ableitungsschritten eine Mutation an einer Produktion vornehmen.
[Ja97] [Ho03] [HLP01] [HP01a] [HP01b] [HP01c] [HP02] [KTV99a] [KTV99b] [Hol01]

2.6.2 Mutation von L-Systemen unter Genetischer Programmierung

Bei der GP werden die L-Systeme als Bäume dargestellt, siehe Abbildung 2.37.
Der Baum ist so organisiert, dass von der Wurzel aus das Axiom und die Produktionen abgehen. Der Knoten *Axiom* enthält als Blätter den Ausdruck des eigentlichen Axioms. Der Knoten *Produktionen* verzweigt in alle Produktionen, die das L-System hat. Eine einzelne Produktion wird wiederum in eine *linke Seite* (auch *predecessor* genannt) und eine *rechte Seite* (auch *successor* genannt) aufgeteilt. Die *linke Seite* enthält als Blatt das Symbol und seine Parameterliste. Die *rechte Seite* geht in eine *Sequenz* über. Um die GP effektiv bei L-Systemen anzuwenden, ist die Strukturierung der rechten Seite der Produktion im Baum sehr wichtig. Für die GP ist eine zu flache Hierarchie nicht sinnvoll, wenn man auch Teilbäume manipulieren möchte. Daher gibt es so genannte Sequenzen, die der rechten Seite eine Hierarchie aufzwingen sollen. Eine Sequenz besteht wieder aus *n* Blättern (in Abbildung 2.37 ist n = 3). Sollte die rechte Seite nur n Symbole enthalten, wäre die Verzweigung der Symbole hier beendet und die Blätter wären mit den Symbolen gefüllt. Sind mehrere Symbole vorhanden, muss ein Blatt durch eine weitere Sequenz ersetzt werden, bis alle Symbole einem Blatt zugeordnet sind. Bei der GP dürfen eckige Klammern nur paarweise auftauchen. Sind trotzdem ein paar eckige Klammern in der Produktion vorhanden, so wird der Inhalt der eckigen Klammern in einem Teilbaum dargestellt. Der Knoten wird mit Stack bezeichnet und erzeugt die gleiche Anzahl an Blättern wie die Sequenz.

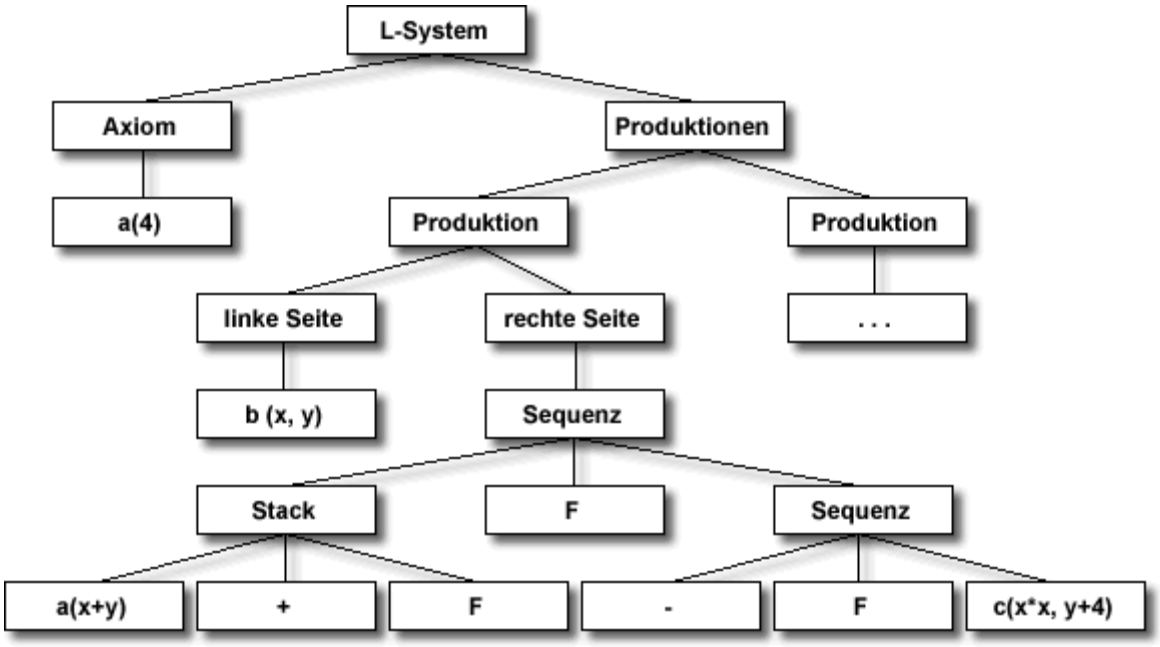


Abbildung 2.38: Ein L-System als Baum dargestellt

In einem L-System-Baum können nur die Blätter und die Knoten *Stack* und *Sequenz* eine Mutation erleben. An Mutationsmöglichkeiten bietet GP für L-Systeme sieben Möglichkeiten an.

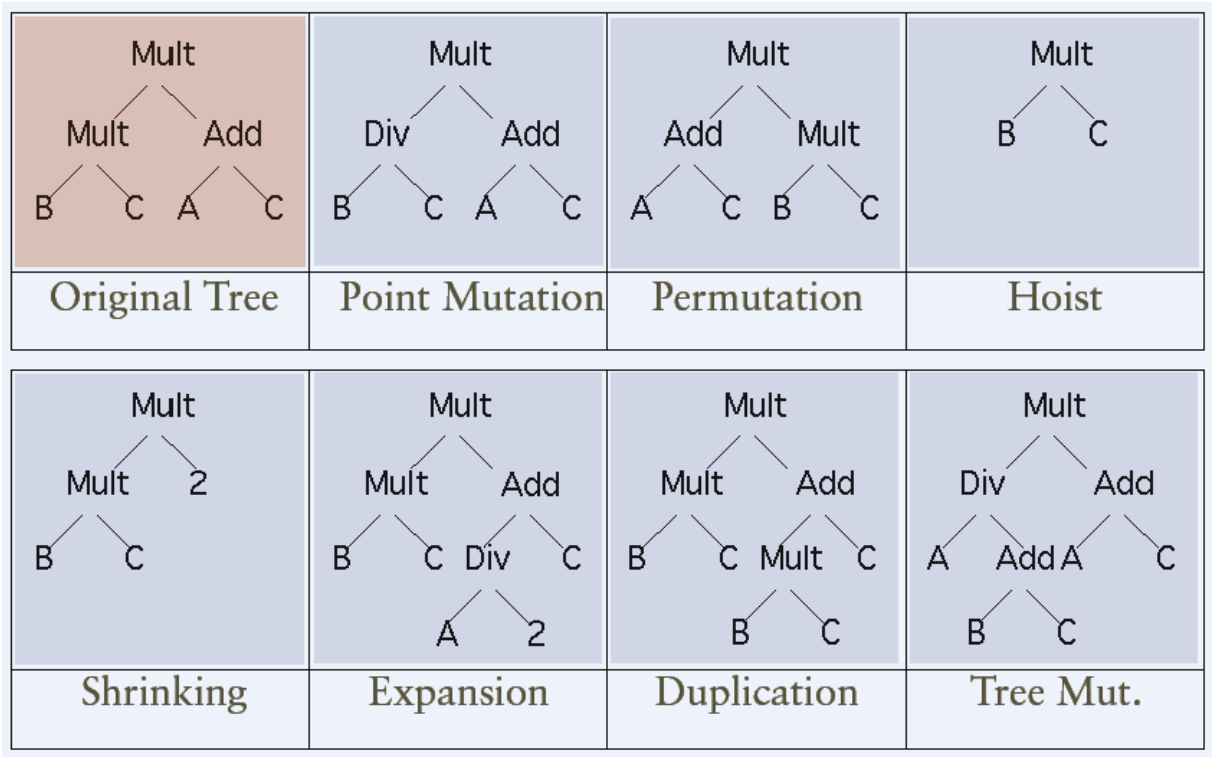


Abbildung 2.39: Mutationsmöglichkeiten unter GP [Ja03]

Diese Möglichkeiten werden hier kurz erläutert.

- Point Mutation
  Es wird ein einzelnes Blatt mutiert.
- Permutation
  Zwei Teilbäume werden vertauscht.

- Hoist
  Dem nächst höheren Knoten werden die Teilbäume abgeschnitten und nur eine Anzahl der Blätter der Teilbäume seinen Blättern übergeben.
- Shrinking
  Ein Teilbaum wird durch ein Blatt ersetzt.
- Expansion
  Ein Blatt wird durch einen neu generierten Teilbaum ersetzt.
- Duplication
  Ein bestehender Teilbaum wird dupliziert und an Stelle eines Blattes hinzugefügt.
- Tree Mutation
  Die ursprünglichen Blätter bleiben erhalten, nur zwischen den Blättern und den ursprünglichen Knoten wird ein neues Stück Teilbaum hinzugefügt. Dabei kann der ursprüngliche Knoten selbst auch mutiert werden.

Wie die GA lassen sich auch die Mutationsarten in die bekannten Mutations-Gruppen einordnen. Die Point-Mutation zählt zu den Genmutationen und alle anderen zu den Chromosomenmutationen. Da bei der GP ein L-System das Genom und Chromosom darstellt, ist das L-System ein haploides Chromosom. Deshalb sind Genommutation ausgeschlossen.
[Ja03] [Ja97] [Ja95] [Ban03] [BNKF98] [Qu02]

2.6.3 Mutation der Interpretation

Eine weitere interessante Möglichkeit der Mutation ist das Verändern der graphischen Interpretation. Bei der *Turtle-Interpretation* ist es möglich, die Bedeutung von zwei Symbolen zu vertauschen. So könnte ein F die Bedeutung eines + bekommen und umgekehrt. Alternativ könnten ganz andere Symbole die Bedeutung der bestehenden graphischen Interpretation übernehmen.
Diese Mutationsmöglichkeit setzt aber voraus, dass das Programm für die graphische Interpretation so flexibel ist, dass es notfalls von selbst noch fehlende eckige Klammern hinzufügt. Außerdem müsste die Evolution, wenn sie auf der genetischen Programmierung basiert, auch die neuen Symbole für den Stack berücksichtigen und den Baum entsprechend ausrichten.

2.7 Applikationsvision

Im Folgenden sollen einige Anwendungsmöglichkeiten zu L-Systemen und deren Evolution vorgestellt werden. Es wird eine grobe Struktur präsentiert sowie die Möglichkeiten oder Ideen, die mit dieser Struktur verbunden sind. Praktischer Bestandteil diese Arbeit ist die Bildgenerierung aus der Grammatik. Zum Evolutionsteil steuert diese Arbeit die Möglichkeiten der Mutation von L-Systemen bei. Soweit es den Anwendungsteil betrifft werden im Wesentlichen Entwürfe angerissen, wie der Anwendungsteil aussehen könnte. Außerdem wird erläutert, wozu solch ein System gebaut wird.

Wie in Abbildung 2.39 zu sehen ist, besteht das System aus drei Teilen. Zwei dieser Teile bilden das Grundgerüst. Die eigentliche Anwendung des Systems ist unabhängig vom Grundgerüst.

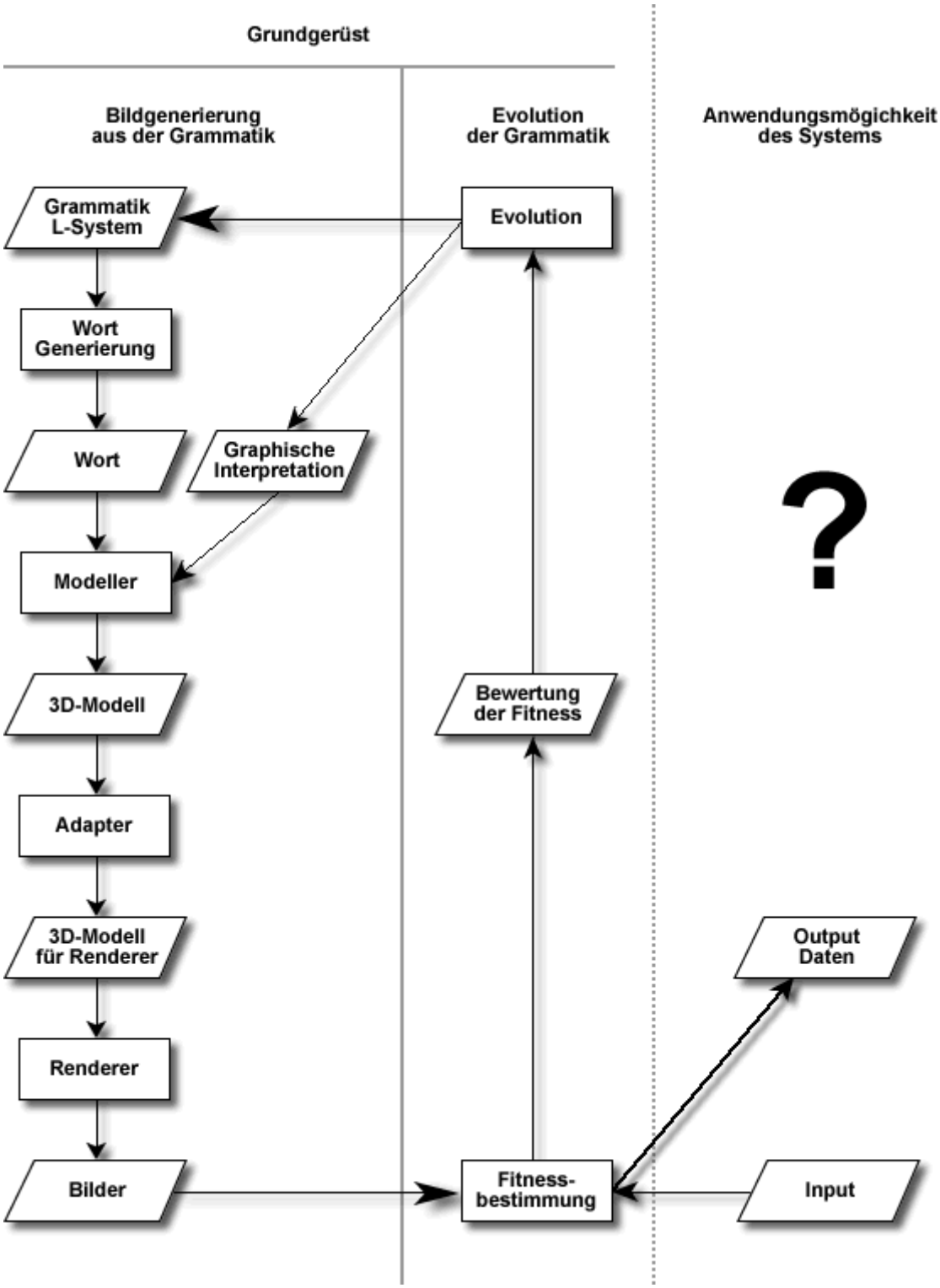


Abbildung 2.40: Grobe Struktur des Systems

2.7.1 Das Grundgerüst

Das Grundgerüst wird in zwei Hälften unterteilt, in die Bildgenerierung aus einem vorgegebenen L-System und in die Evolution.
Die Bildgenerierung bekommt als Eingabedaten ein L-System und die graphische Interpretation. Dabei basiert die graphische Interpretation auf der *Turtle-Interpretation*. Die Grammatik dient als Eingabe für ein Programm zur Ableitung eines Wortes. Das generierte Wort wird mit der graphischen Interpretation einem Modeller übergeben. Der Modeller selbst verhält sich wie die Schildkröte im dreidimensionalen Raum und erzeugt ein 3D-Modell, das sich aus den Eingabedaten herleitet. Im Anschluss wird das allgemeine 3D-Modell in ein spezielles 3D-Modell für ein bestimmtes Ray-Tracing-Programm überführt. Diese Arbeit übernimmt ein Adapter. Dadurch ist es möglich, ein beliebiges Ray-Tracing-Programm zu nutzen. Ausgetauscht werden muss lediglich der Adapter. Sollte aber das allgemeine 3D-Modell schon von einem Ray-Tracing-Programm verstanden werden, ist der Schritt über den Adapter nicht nötig. Im letzten Verarbeitungsschritt zum Bild wird dem Ray-Tracing-Programm das 3D-Modell übergeben, aus dem das Bild erzeugt werden soll. Diesen Vorgang nennt man Rendering, deshalb werden Ray-Tracing-Programm auch kurz Renderer genannt. Gegebenenfalls wird vom Renderer nicht nur ein Bild erzeugt, sondern mehrere Bilder des gleichen Objektes, nur aus verschiedenen Kameraeinstellungen.
Im ersten Schritt der Evolution bekommt das Programm zur Fitnessbestimmung die erzeugten Bilder aus der Grammatik. Die Fitnessbestimmung hängt größtenteils von der Eingabe des Anwendungsteils ab. In Abhängigkeit davon, was von der Anwendungsseite zur Fitnessbestimmung übergeben wird, erzeugt die Fitnessbestimmung eine Ausgabe. Diese Ausgabe kann Verschiedenes enthalten, dies wird bei den Anwendungsmöglichkeiten näher erläutert. Gleiches gilt auch für die Spezifizierung der Eingabe. Nachdem die Fitness bestimmt ist, werden die Grammatik und die graphische Interpretation einer Evolution unterzogen. Beides bringt wiederum eine neue Grammatik und eine neue graphische Interpretation zum Vorschein, die wiederum direkt dem Bilderzeugungsprozess übergeben wird. Das beste Individuum sollte dabei immer zwischengespeichert werden, damit es nicht verloren geht. Falls die graphische Interpretation einer Mutation unterzogen wurde kann es passieren, dass die neue Grammatik viel weiter von der Lösung entfernt ist. Sollte sich das Ergebnis nach mehreren Evolutionsschritten nicht verbessern, kann eine Kopie des besten Individuums und seiner Interpretation der Evolution übergeben werden. Die andere Grammatik und Interpretation können dann verworfen werden.

2.7.2 Die Anwendungsmöglichkeiten

Applikation: Imogene
Als Vorbild dient das Programm Imogene [[Imo]], das auch auf der CD-ROM beiliegt. Imogene erzeugt Bilder, die manchmal eine Struktur erkennen lassen, manchmal aber auch wie ein *„Ameisenbild“* aussehen. Der Anwender kann aus neun Bildern das auswählen, das ihm am besten gefällt. Anhand des ausgesuchten Bildes wird eine neue Generation von Bildern erzeugt.
Eine Anwendungsmöglichkeit wäre ein Programm in der Art von Imogene. Dabei könnte man als Initialpopulation neu generierte L-Systeme verwenden oder auf eine bestehende Basis von L-Systemen aufsetzen. Diese Neugenerierung kann zu einer Art von abstraktem Bild führen. Die bestehende Basis von L-Systemen stellt eine Menge dar, aus der einige L-Systeme gewählt werden, die dann die Anfangspopulation darstellen. So könnte es Mengen von L-Systemen geben, die Bäume, Pflanzen, Häuser oder Gebirge darstellen.

Der Anwender bestimmt dabei immer noch, welches erzeugte Bild ihm am besten gefällt. Die Basis für die Evolution bildet das L-System, das dahinter steht. Aus dem L-System werden nun durch die Evolution so viele neue Individuen erzeugt, bis die Population vollständig ist. Erneut kann der Anwender aus der neuen Population seinen subjektiven Favoriten auswählen. Diese Schleife wird so lange fortgeführt, bis der Anwender kein Interesse mehr an der Fortsetzung hat. Es liegt nahe, dass auch nur eine Grammatik für die Evolution zur Verfügung steht, wenn nur ein Bild ausgewählt wird. Damit wäre nur die Mutation als Evolution sinnvoll. Eine Reproduktion wäre aus der Sicht des Anwenders wahrscheinlich nicht wünschenswert, da er nicht das gleiche Bild erneut erwartet.
Durch die Auswahl des Bildes legt der Anwender auch die Eingabe für die Fitnessbestimmung fest und damit auch gleichzeitig die Bestimmung des Individuums, das einer erneuten Evolution unterzogen werden soll. Als Ausgabe erhält der Anwender die generierten Bilder.

Applikation: Alternatives L-System
Das Einsatzgebiet von GA und GP sind Probleme, die nicht mit herkömmlichen Lösungsmethoden lösbar sind. Man könnte nun folgendes Problem definieren.

> Auf der Basis eines L-Systems wird ein Bild erzeugt. Nun soll über ein Programm das L-System gefunden werden, das solch ein Bild erzeugt. Das L-System, welches das Bild erzeugt hat, steht zur Findung nicht zur Verfügung.

Interessant ist, ob dieses L-System gefunden wird oder ein anderes L-System, welches das gleiche Bild erzeugt. Dabei kann das andere L-System vielleicht mehr oder weniger Produktionen und kürzere oder längere Produktionen aufweisen. Wenn ein anderes L-System gefunden wurde, das auch das Bild erzeugt, ist weiterhin interessant, ob das Wort das gleiche ist oder nicht.
In diesem Zusammenhang spricht man auch vom *Inferenzproblem* für L-Systeme.

> *„Das* Inferenzproblem *für L-Systeme besteht darin, ein geeignetes Axiom und entsprechende Produktionen zu finden, so daß eine gegebene Struktur oder ein bestimmter Wachstumsprozeß nachgebildet werden.“*
> [Ja97]

Als Eingabe für die Fitnessbestimmung gibt es einmal das Bild, für das ein L-System gesucht wird, und ein Bild, von dem/den L-System/en das/die nach dem L-System sucht/suchen. Über die Ähnlichkeit der Bilder lässt sich dann die Fitness bestimmen.
Als Ausgabe erhält der Anwender entweder das gefundene L-System, welches das Bild erzeugen kann, oder das L-System, das dem Ergebnis am nächsten kommt. Dabei sollten die Fitnessergebnisse dem Anwender auch zukommen.
Interessant ist auch die Frage, wie viel Zeit für die Findung eines L-Systems benötigt wird, das möglichst nahe an die Lösung kommt.

Applikation: Effektive Speicherung von Pflanzen
Prusinkiewicz zeigt in seinen Veröffentlichungen [PL90] [Pr86] [Pr93] [PJM94] [PHM] [PHHM96a] [PHHM96b] [PHHM95] [BPFGK03], dass alle erdenklichen Pflanzen mit L-Systemen für die Computergrafik beschrieben werden können. Dieser Prozess könnte auch umdreht werden indem versucht wird, von einem Bild ein L-System zu generieren, das möglichst nahe an das Originalbild der Pflanze kommt. Dadurch könnte eine Pflanze sehr effektiv gespeichert werden.

Als Eingabe für die Fitnessbestimmung steht das Foto des Objektes zur Verfügung sowie die generierten Bilder der L-Systeme. Über die Ähnlichkeit kann dann die Fitness bestimmt werden. Als Ausgabe erhält man das L-System, das dem Foto am nächsten kommt.

Applikation: Pflanzenerkennung
Mittlerweile gibt es eine große Menge von L-Systemen, die reale Pflanzen erzeugen. Ein Teil dieser L-Systeme könnte in einer Datenbank abgelegt werden. Diese Datenbank würde eine Basis von repräsentativen Pflanzen bilden. Diese L-Systeme könnten nun aus der Datenbank entnommen werden und mit Hilfe von GA oder GP einer Evolution unterzogen werden, um auf einem Foto eine Pflanze zu erkennen. Wird beispielsweise das Foto einer Rose aufgenommen und in der Datenbank liegt auch ein repräsentatives Modell einer Rose, dann wird vom Programm zunächst dieses Modell genommen. Nach zahlreichen Evolutionsschritten entsteht so eine Rose, die dem Foto sehr nahe kommt. Das Programm entschließt, dass auf dem Foto eine Rose ist und teilt das dem Anwender mit.
Dieser grobe Überblick soll noch ein etwas spezifiziert werden. Als Eingabe erhält der Evolutionsprozess das Foto einer Pflanze sowie die Datenbank mit den repräsentativen Pflanzen (Eine Auswahl repräsentativer Modelle zu den wichtigsten Bäumen ist in [De03] zu finden). Das Programm entnimmt nun ein Modell aus der Datenbank und bestimmt die Ähnlichkeit des Modells mit dem Foto. Diese Ähnlichkeit des Modells zum Foto ist die Fitnessbestimmung. Nun durchläuft das Modell eine bestimmte, vorher festgelegte, Anzahl von Evolutionsschritten. Nach jedem Evolutionsschritt wird die Fitness des neuen Modells bestimmt. Sollte eine bestimmte Ähnlichkeitsschwelle überschritten werden, so erhält die Pflanze auf dem Foto die Bezeichnung des Modells. Bevor festgelegt wird, dass die Pflanze eine Alternative zum Modell ist, muss eine Ähnlichkeitsprüfung zwischen dem Modell und dem generierten Bild vorgenommen werden. Damit soll sichergestellt werden, dass durch die Evolutionsschritte aus einem ursprünglichen Modell, wie zum Beispiel aus einer Rose, nicht ein vollkommen anderes Objekt entsteht, beispielsweise ein Haus. Konnte diese Schwelle mit dem ersten Modell nicht überschritten werden, dann wird ein anderes Modell aus der Datenbank entnommen. Das geschieht so lange, bis alle Modelle geprüft sind.
Wenn kein Modell der Pflanze auf dem Foto entspricht, dann könnte die Applikation versuchen, ein neues Modell aufzustellen, um die neue Pflanze in die Datenbank aufzunehmen. Eine Einstiegshilfe könnte das erste neue Modell sein, was bei den vorherigen Durchläufen die beste Fitness aufweisen konnte. Natürlich kann auch ein komplett neues Modell per Zufall generiert und der Evolution unterworfen werden.
Wurde ein entsprechendes Modell für die neue Pflanze gefunden, könnte das Wort, das die Pflanze erzeugt, einem intelligenten Analyseunterprogramm gereicht werden. Dieses könnte versuchen, aus der Struktur, die das Wort repräsentiert, eine Zuordnung zu einer groben Pflanzengattung machen, beispielsweise Blume oder Baum. Dazu kann das Analyseunterprogramm auf Merkmale der verschiedenen Pflanzengattungen zurückgreifen, bei einem Baum beispielsweise auf die Merkmale sehr hoher Stamm und eine große Anzahl von Verzweigungen. Repräsentiert werden könnte das Analyseunterprogramm zum Beispiel durch ein Neuronales Netz.

3 Programme, die L-Systeme simulieren/interpretieren

In diesem Kapitel wird ein Überblick über die vorhandenen Applikationen zu L-Systemen gegeben und ihre unterschiedliche Eigenschaften zu verdeutlichen.

3.1 Lparser

Das Softwarepaket Lparser [[LP4]] liegt in der Version 4 vor und stammt hauptsächlich vom Autor Laurens Lapre. Es ist zwar schon älter (1995), aber sehr verbreitet bei Windows-Anwendern.
Lparser besteht stark vereinfacht aus drei Komponenten, aus einem Programm, das ein Wort aus einem L-System generiert und als 3D-Modell abspeichert, aus einem Programm zum Betrachten des 3D-Modells als Drahtgittermodell und aus einem Programm zum Konvertieren des 3D-Modells in eine POV-Ray Datei. Das Konvertierungsprogramm wurde von Cees van der Mark jr. geschrieben.
Das Programm zum Erzeugen eines 3D-Modells nennt sich Lparser und ist ein Kommandozeilenprogramm. Dieses bekommt als Eingabe eine Datei (*.LS), in der ein L-System definiert ist. Dabei unterstützt Lparser PDB0L-Systeme, lediglich die Turtle-Kommandos können parametrisiert werden. Neben dem L-System selbst können auch Optionen über die Kommandozeilenargumente übergeben werden. Außer dem POV-Ray Format werden die Formate DXF (AutoCAD), RAW, WRL (VRML), RDF (RenderStar) und BLB (Blob Sculptor) unterstützt. Eine weitere Eigenschaft ist die Möglichkeit, ein L-System mutieren zu lassen, wobei als Parameter nur die Anzahl der Mutationsdurchläufe angegeben werden kann. Die Mutation lässt die einzelnen Produktionen mutieren, entspricht also dem Vorbild von GA (vgl. Kapitel 2.3).
Das Anzeigeprogramm LViewer bietet die Möglichkeit, sich das erzeugte 3D-Modell anzeigen zu lassen. Angezeigt wird ein Drahtgittermodell, das rotiert und skaliert werden kann. Die Änderungen werden dabei nicht im POV-Ray 3D-Modell gespeichert. Lediglich für RenderStar 3D-Modelle können die Einstellungen gespeichert werden. Für alle andere 3D-Modelle kann eine INFO.TXT Datei generieren werden, mit den Informationen über die Kameraeinstellungen.
Das Konvertierungsprogramm heißt LV2POVID. Dabei wird nicht nur POV-Ray unterstützt sondern auch ViVid. Das Programm erzeugt für POV-Ray eine Szene, die kompatibel zur Version 2.X ist. Die einzige Änderung, die im POV-Ray Editor ausgeführt werden muss, um zur Version 3.X kompatibel zu sein, ist, dass die declare-Anweisungen ein Semikolon benötigen, weil diese eine Aufzählung darstellen. Ansonsten kann die Szene sofort gerendert werden. Neben der LPAR2POV.POV Datei erzeugt das Programm eine Include-Datei, LPAR2POV.INC, in der die Daten der 3D-Primitiven beschrieben sind.

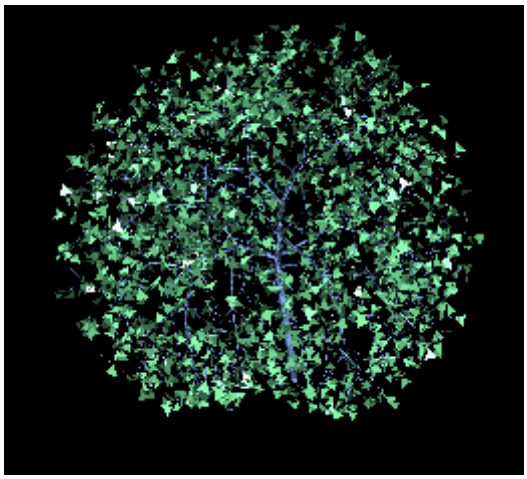


Abbildung 3.1: Baum, erzeugt von Lparser und gerendert mit POV-Ray

3.2 L-System

Das Programm L-System wurde von Timothy C. Perz [[LS4]] 1999 entwickelt und liegt in der Version 4.01 vor. L-System basiert dabei auf Lparser mit dem Unterschied, dass es eine vollständige Windows-Applikation mit einer graphischen Oberfläche ist. So bindet L-System alle einzelnen Komponenten zu einem Programm zusammen und erhöht damit die Transparenz und Benutzerfreundlichkeit. Weil es auf Lparser basiert, sind alle *.LS, die unter Lparser laufen, auch unter L-System ausführbar.
Gegenüber Lparser enthält L-System einige Erweiterungen. Eine wichtige ist die Unterstützung von parametrisierten L-Systemen. Damit können auch komplexe Modelle erzeugt werden.
Außerdem wurden die Mutationsmöglichkeiten gegenüber Lparser stark erweitert. Für die Mutation wurde ein Mutator entwickelt. Über ihn lassen sich verschiedene Mutationsarten regeln (siehe Kapitel 2.5.1). Bei der Anwendung entsteht jedoch der Eindruck, als ob sowohl bei Lparser als auch bei L-System die Mutation noch nicht ausgereift ist, weil die Ergebnisse deterministisch sind bzw. nicht stark genug hervorkommen. Wird beispielsweise das gleiche L-System zweimal mit derselben Gewichtung bei den Mutationsfaktoren mutiert, so wirkt das Ergebnis nahezu identisch.
Eine andere Erweiterung von L-System ist der Random-Generator. Über eine Vielzahl von Parametern, lassen sich willkürliche L-Systeme generieren. Erzeugt diese generierte Grammatik ein langes Wort, dann stürzt L-System ab.
Das Anzeigeprogramm von L-System ist gegenüber LViewer in einem Teil schwächer. So lassen sich die 3D-Modelle nur im DXF-Format speichern.
Die Grafiken lassen sich skalieren und rotieren. Dabei wird die Kameraeinstellung mitgespeichert. Außerdem kann neben dem Drahtgittermodell auch ein komplettes 3D-Modell mit und ohne Texturen angezeigt werden. Die Texturen und Farben können im Anzeigeprogramm verändert werden. Vor der Anzeige wird das Wort daraufhin überprüft, ob es überhaupt interpretierbar ist.

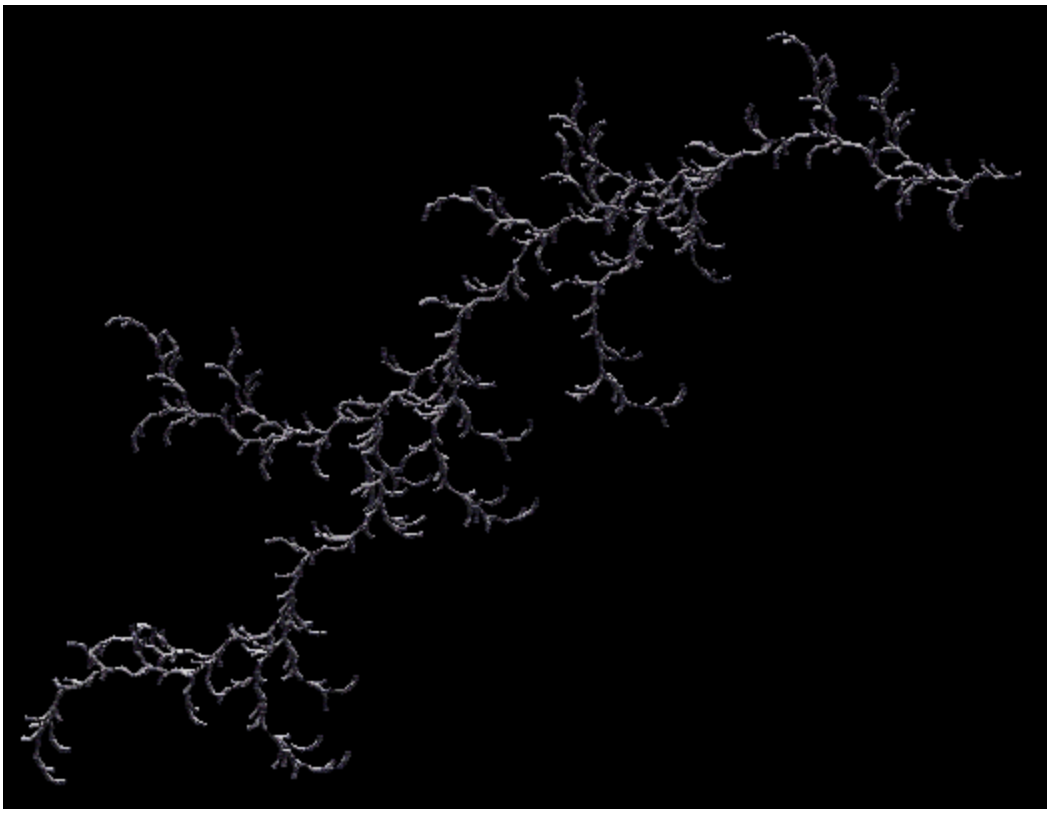


Abbildung 3.2: Beispiel für eine 3D-Grafik von L-System

3.3 RayTraced Evolution

RayTraced Evolution wurde von Nguyen Duc Cuong an der TU Ilmenau entwickelt [[RTE]]. Die vorliegende Version 1.1 wurde 1997 erstellt.
Das Programm erzeugt aus einer Beschreibungs-Datei (*.RTE) ein 3D-Modell für einen speziellen Renderer und speichert es in einer Datei ab. Dabei wird bei der Ausgabe auf einen Treiber für den entsprechenden Renderer zurückgegriffen. So kann durch Austausch des Treibers das 3D-Modell für einen anderen Renderer benutzt werden. Als Standard ist eine POV-Ray Unterstützung integriert. Um eine Kollision der Turtle-Bewegungen zu vermeiden, wurde der GX/GENETIC Ray-Tracing Kernel eingebaut. Sollte die Schildkröte irgendwo kollidieren, können verschiedene Aktionen ausgeführt werden, die vom „Sterben“ bis zur „Reinkarnation“ reichen. Die generierte Szene kann ohne Veränderung direkt mit POV-Ray gerendert werden.
Eine weitere mächtige Eigenschaft von RayTraced Evolution ist die Beschreibungssprache für eine Szene. Sie ist in zwei Sprachen unterteilt, in eine C-ähnliche Hochsprache und in die L-Systeme. Seitens der Hochsprache werden dabei auch Datentypen wie Vektoren und Matrizen unterstützt sowie die meisten Operatoren, die aus C bekannt sind. Viele Bibliotheken unterstützen dabei die Hochsprache, wie zum Beispiel eine Mathematik-Bibliothek oder eine reine Bezier-Bibliothek. Auch Unterprogramme werden unterstützt.
Neben der mächtigen Sprache werden auch eine Vielzahl von L-Systemen unterstützt. RayTraced Evolution unterstützt parametrisierte und stochastische L-Systeme, T0L- und B0L-Systeme sowie Timed L-Systeme für Animationen. Alle L-Systeme müssen jedoch deterministisch und propagierend sein.
Das Programm selbst ist ein Kommandozeilenprogramm und hat keine weitere graphische Oberfläche.

Der einzige Nachteil ist, dass nur eine spärliche Dokumentation existiert und ansonsten keine weitere Unterstützung.

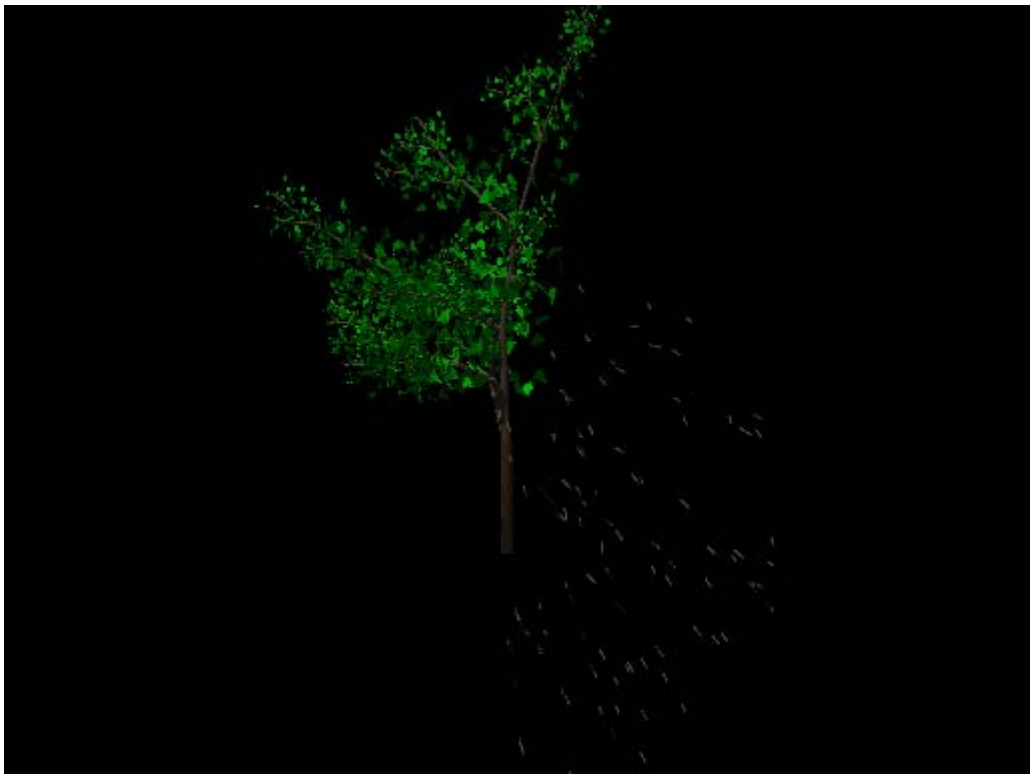


Abbildung 3.3: 3D-Modell erzeugt von RayTraced Evolution

3.4 LinSys3D

Als Nächstes soll das von Andrea Esuli entwickelte Programm LinSys3D [[L3D]] vorgestellt werden. Die derzeitige Version ist 1.2, sie stammt aus dem Jahre 2001.
LinSys3D ist eine Windows-Anwendung mit einer graphischen Oberfläche ohne Kommandozeilen-Unterstützung.
Die Applikation unterstützt eine Reihe von L-Systemen, so zum Beispiel stochastische, parametrisierte und kontextsensitive L-Systeme. Alle L-Systeme müssen deterministisch und propagierend sein. Ein Unterschied zu vielen anderen Programmen ist der Ablauf, also wie die Pflanze synthetisiert wird.
Im ersten Schritt wird das L-Schema angegeben. Dazu wird als Erstes in eckigen Klammern das Alphabet aufgelistet. Die Produktionen selbst werden in geschweiften Klammern aufgelistet, dabei gilt folgende Syntax:

{Symbol, linker Kontext, rechter Kontext, [Bedingung] :
<Wahrscheinlichkeit1> Produktion1, <Wahrscheinlichkeit2> Produktion2,
..., <WahrscheinlichkeitN> ProduktionN}

Nachdem das L-Schema definiert ist, wird ein Syntaxcheck ausgeführt. Sollte das L-Schema einen Fehler aufweisen, muss dieser erst behoben werden, bevor die Verarbeitung fortgesetzt werden kann.

Als Nächstes werden vordefinierte Drahtgittermodelle geladen, zum Beispiel für ein Blatt oder einen ganzen Blütenkranz. Dabei werden die Modelle aus einer Text-RAW Datei gelesen. Das Modellieren der Komponenten geschieht entweder per Hand oder mit Hilfe von Programmen wie Crossroads.
Im dritten Schritt wird den Symbolen aus dem Alphabet eine Eigenschaft zugewiesen. So kann ein Symbol als Signal definiert werden, damit hat es keine graphische Bedeutung und wird nur für das Wachstum der Pflanze benutzt. Ein Symbol kann auch eine Raumtransformation darstellen, zum Beispiel rotieren oder skalieren. Außerdem lassen sich durch Symbole zwei verschiedene Typen von Grafikobjekten darstellen, ein statisches und ein parametrisiertes. Zu jedem Grafikobjekt muss das Drahtgittermodell aus dem vorherigen Schritt angegeben werden.
Im vorletzten Schritt wird das L-Schema zum L-System vervollständigt und das zu interpretierende Wort generiert. Es wird das Axiom angegeben und die Anzahl der Ableitungsschritte. Zuletzt wird das Wort mit Hilfe von OpenGL graphisch dargestellt. Das 3D-Modell kann entweder als BMP-Grafik oder als POV-Ray Datei gespeichert werden.

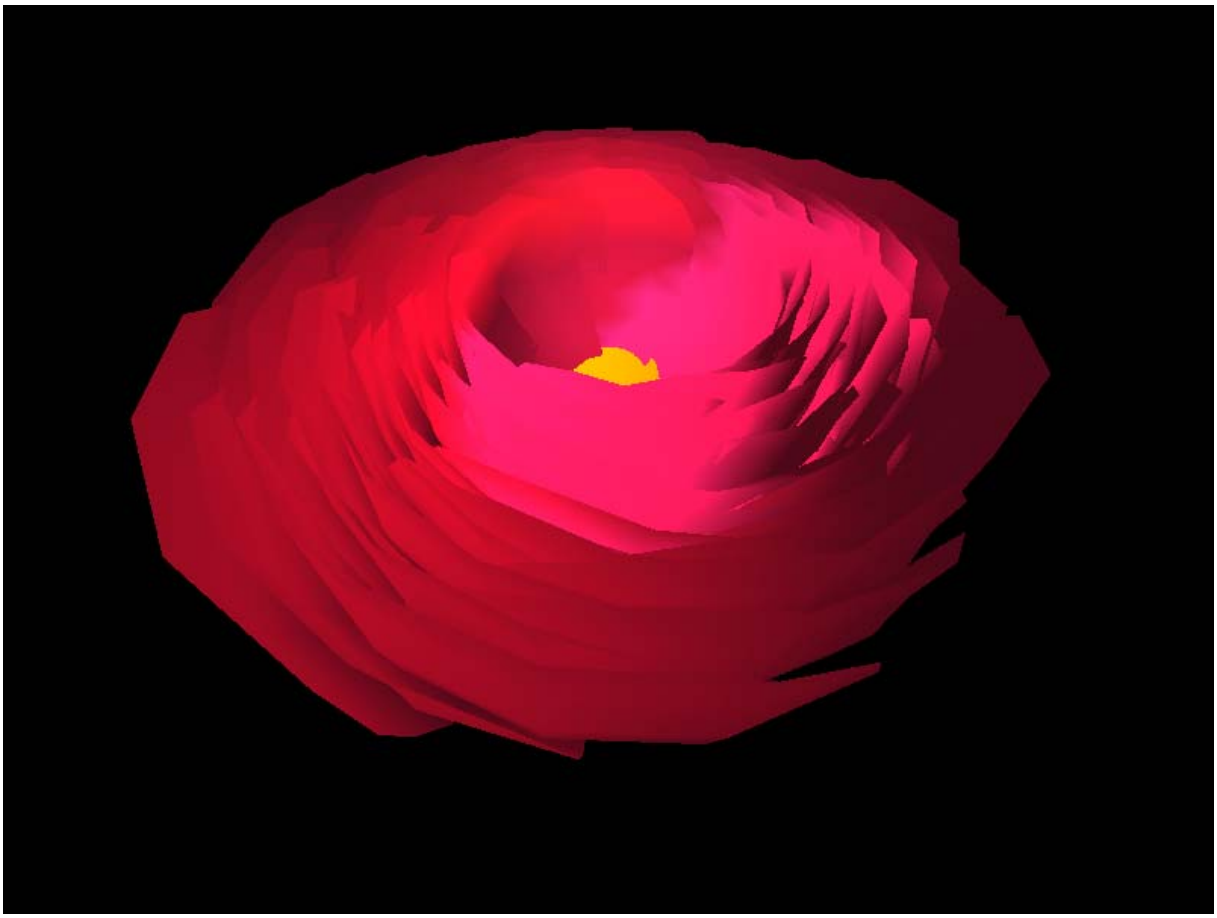


Abbildung 3.4: Eine Rosenblüte erstellt von LinSys3D

3.5 Zusammenfassung und Übersicht

Eine tabellarische Zusammenfassung der Programme ist in den Tabellen 3.1-a und -b zu finden.
Ein wichtiger Aspekt von Lparser ist, dass es eine große Verbreitung aufweisen kann. Dem entsprechend existieren unzählige Tools für Lparser. Einige sollen kurz genannt werden.

- *PlanD*
  Ist eine kleine Entwicklungsumgebung für den Lparser. Alle Einzelkomponenten von Lparser sind in einer graphischen Oberfläche vereint. Der Viewer zeigt ein 3D-Modell der aktuellen Grammatik an. Wird etwas verändert, wird das 3D-Modell sofort angepasst. [[Plan]]
- *Lparser Update*
  Da Lparser freie Quelldateien hat, bietet es sich an, auch diese zu erweitern. Ein Autor hat Lparser um parametrisierte L-Systeme erweitert. [[LP5]]
- *Lsys32*
  Ist eine graphische Oberfläche, in der ein L-System definiert werden kann. Zusätzlich bietet das Programm an, viele Parameter von Lparser über die Oberfläche zu steuern. [[L32]]

Im Folgenden sollen einige weitere Programme zu L-Systemen kurz vorgestellt werden.

GROGRA:
GROGRA ist eine Software, deren Hauptziel nicht nur die Visualisierung von L-Systemen ist. Das Programm wurde entwickelt, um das Wachstum von Gehölzen zu simulieren und diese zu analysieren, zum Beispiel den Saftfluss im Baum oder die Fotosynthese. Die Software wurde zur Waldökosystemforschung entwickelt.[3] [[GRO]]

Fractint:
Ein schon älteres, aber bekanntes Programm ist Fractint. Mit der Software lässt sich fast jedes erdenkliche Fraktal erzeugen. Einfache PDB0L-Systeme werden auch unterstützt. Die graphische Ausgabe ist normalerweise zweidimensional, kann aber auf die dritte Ebene erweitert werden. Es existiert auch eine Exportfunktion für Renderer. Für den POV-Ray existiert nur ein RAW-Format, welches erst mit einem weiteren Tool namens RAW2POV umgewandelt werden muss. Dieses Tool wurde aber von den Entwicklern von POV-Ray eingestellt und ist nicht mehr verfügbar. [[Fra]]

Lworld:
Lworld ist ein Programm zur Animation von L-Systemen mit OpenGL. Das Programm wurde an der Universität Zürich von Herrn Hansrudi Noser entwickelt. [[Lwo]]

LS-SketchBook:
LS-SketchBook unterstützt derzeitig parametrisierte, stochastische und kontextsensitive L-Systeme. Im Vergleich zu anderen Programmen unterstützt es auch nicht propagierende L-Systeme. Die Ausgabe erfolgt mit OpenGL. Außerdem gibt LS-SketchBook den kompletten Wachstumsprozess graphisch aus. Dieser kann als Animation mitgeschnitten werden. [[LSB]]

[3] Hier soll Herrn Prof. Dr. Winfried Kurth (BTU Cottbus) gedankt werden, dass er eine Kopie von GROGRA für diese Arbeit zur Verfügung gestellt hat.

L-studio:
L-studio ist die Windows Version des Virtual Laboratory. L-studio wurde, wie das Virtual Laboratory, von Prusinkiewicz entwickelt. Das Programm ist sehr mächtig, aber teilweise schwer zu bedienen. Es lassen sich parametrisiert, stochastische und kontextsensitive L-Systeme nutzen und visualisieren. L-studio selbst ist nur ein Editor, innerhalb dessen das L-System, die Parameter, Farben, Oberflächen, Konturen und viele weitere Optionen beschrieben werden können. Die Visualisierung wird von einem externen Programm namens Cpfg vorgenommen. In Cpfg lassen sich die 3D-Modelle als Bild (BMP, TGA, ...), RayShade Datei, PostScript, String, Gls, View Volume, Inventor oder OBJ-Datei speichern. Das Programm liegt der CD-ROM nicht mit bei, weil es nur als Demo-Version herunterzuladen ist, die in einem Monat abläuft. Die Demo-Version lässt sich unter http://www.cpsc.ucalgary.ca/Research/bmv/lstudio/ herunterladen. [[LSt]]

Abbildung 3.5: Eine Lilie erstellt mit L-studio

Name	Version / Jahr	Autor / Homepage	Sourcen verfügbar	Unterstützt folgende L-Systeme	Kommandozeile / GUI
Lparser	4.0 /1995	Laurens Lapre http://home.wanadoo.nl/laurens.lapre/	Ja / in C	PDB0L-Systeme	Ja / nur Viewer
L-System	4.01 /1999	Timothy C. Perz http://www.geocities.com/tperz/L4Home.htm	Nein	Parametrisierte PDB0L-Systeme	Nein / Ja
RayTraced Evolution	1.1 / 1997	Nguyen Duc Cuong http://www.stud.tu-ilmenau.de/~juhu/GX/RTEvol/	Nein	Parametrisierte, stochastische, Timed, Table, PDB0L-Systeme	Ja / Nein
LinSys3D	1.2 / 2001	Andrea Esuli http://web.tiscali.it/esuli/LinSys3d/	Nein	Parametrisierte, stochastische, kontextsensitive, PDB0L-Systeme	Nein / Ja

Tabelle 3.1-a: Übersicht über die Software zu L-Systemen

Viewer	Formate	Mutation	Vorteile	Nachteile
Externer Viewer, Drahtgittermodell	DXF, RAW, POV, WRL, BLB, RDF	Anzahl der Mutationsschritte	Viele Tools, Sourcen, viele Formate, breite Unterstützung	Nur PDB0L-Systeme, viele Einzelprogramme, einfache Mutation
Drahtgittermodell, 3D-Modell mit /ohne Texturen, Textur/ Farbe wählbar	DXF, BMP, JPG	Verschiedene einstellbare Parameter	Mutation, Viewer, Random-Generator, GUI	Wenige Formate und läuft nicht stabil
Nein	POV / Treiber abhängig	Nein	Viele L-Systeme, Treiber, mächtige Sprache	Keine Mutation, kein Viewer, keine gute Dokumentation
3D-Modell, Multi-/ Camera-/Fullscreen, mehrere Lichtquellen, OpenGL	BMP, POV	Nein	Viele L-Systeme, Viewer, Komponentenansatz	Keine Mutation, wenige Formate, komplexe Handhabung

Tabelle 3.1-b: Übersicht über die Software zu L-Systemen

4 Entwurf und Implementierung der Fassade

4.1 Der Begriff der Fassade

Mit dem Wort *Fassade* bringt man häufig ein Gebäude in Verbindung, sie bildet sozusagen die Außenhaut eines Hauses. Dabei verdeckt die Fassade die gesamte Komplexität nach innen. So sind Wasserrohre, Elektroleitungen und vieles von außen nicht sichtbar. Durch die Fassade wird einem Außenstehenden der Einstieg /die Einsicht in das Gebäude erschwert. Über die Fassade wird jedoch eine Schnittstelle definiert, über die der Außenstehende sein Anliegen an die Personen im Gebäude übermitteln kann.
Dieses wiederkehrende Muster in der Architektur, engl. Patterns, hat der Architekt Christopher Alexander in seinem Buch „A Pattern Language“ 1977 festgehalten [GHJV96]. Inspiriert von Alexanders Buch und seinen Ideen setzt Erich Gamma diese Pattern in spezifische Entwurfsmuster für Programmierprobleme um. Eines dieser Entwurfsmuster ist die Fassade, sie zählt zur Untergruppe der Strukturmuster. Zweck dieser Strukturmuster ist das Zusammenführen von vielen verschiedenen Objekten zu einer großen Struktur mit einer einheitlichen Schnittstelle. Die Fassade selbst wird in [GHJV96] wie folgt definiert:

> *„Bietet eine einheitliche Schnittstelle zu einer Menge von Schnittstellen eines Subsystems. Die Fassadenklasse definiert eine abstrakte Schnittstelle, welche die Benutzung des Subsystems vereinfacht.“*

Normalerweise werden Entwurfsmuster im objektorientierten Bereich verwendet. Die Fassade in der vorliegenden Diplomarbeit wird aus Performancegründen jedoch in C entwickelt. Die Merkmale der Fassade lassen sich dabei auch ohne die Verwendung einer objektorientierten Sprache realisieren.

4.2 Einführung zum Entwurf

Mit diesem Kapitel beginnt der Praxisteil der Diplomarbeit. Hier liegt der Schwerpunkt auf dem Konzept, dem Entwurf und der Implementierung der Software.
Die Entwicklung der Software unterteilt sich in zwei separate Bereiche. Der erste Teil ist der bereits aus dem Theorieteil bekannte Prozess der Verarbeitung der Grammatik bis zur Bilderzeugung. Dieser theoretische Ablauf ist in Abbildung 2.39 dargestellt. Er wird im Verlauf des Kapitels diskutiert und als Fassade implementiert. Auch auf die Programme, die von der Fassade aus bedient werden, wird noch näher eingegangen. Außerdem wird die Auswahl der Programme begründet und erläutert, welche Probleme mit den jeweiligen Programmen einhergehen. Ziel ist es, das entwickelte Programm in einen Evolutionsprozess einzubetten oder auch in andere Abläufe zu integrieren, die eine Visualisierung von L-Systemen benötigen. Dabei soll die Fassade eine beherrschbare Anzahl von Optionen beherbergen, ohne dass die Übersichtlichkeit verloren geht.
Nachdem im ersten Teil der Softwareentwicklung eine Verarbeitungsschicht erstellt wurde, findet im zweiten Teil die Entwicklung einer Oberflächen-Schicht statt. Das Ziel der Oberflächen-Schicht ist das praktische Arbeiten mit dem Prozess aus dem ersten Teil, nicht die Einbettung in einen automatischen Prozess sein. Die entwickelte graphische Oberfläche bietet die Möglichkeit, L-Systeme komfortabel einzugeben, Optionen für den Prozess zu setzen und den Prozess für die Bilderzeugung ablaufen zu lassen.

Für die Fassade und für die graphische Oberfläche wurden zwei verschiedene Entwicklungsumgebungen und zwei verschiedene Programmiersprachen verwendet. Im Folgenden wird zunächst die Entwicklung der Fassade erläutert. Auf die graphische Oberfläche wird dann in Kapitel 5 eingegangen.
Zuvor soll jedoch erwähnt werden, dass die fertige Fassade Lprocess heißt und die graphische Oberfläche „Visual L“.

4.3 Die Entwicklungsumgebung

Für die Entwicklung der Fassade wird auf das Visual Studio 6.0 von Microsoft zurückgegriffen, denn diese Entwicklungsumgebung bietet eine komfortable und gut strukturierte Umgebung zur Entwicklung von Konsolenapplikationen und einen leistungsfähigen Debugger. Als Programmiersprache wird C gewählt, weil C auf anderen Betriebssystemen sehr verbreitet ist. Außerdem bietet ein in C entwickeltes Programm eine hohe Geschwindigkeit, was gerade bei einem Evolutionsprozess nicht unwesentlich ist, weil dieser insgesamt sehr zeitaufwendig ist. Wobei bei der Entwicklung darauf geachtet wurde, so wenig wie möglich Windows-spezifische Funktionen und Bibliotheken zu verwenden, damit die Möglichkeit besteht, das Programm mit wenig Aufwand auf einem anderen Betriebssystem zum Einsatz zu bringen.

4.4 Modulübersicht

In Kapitel 2.6 wurde schon kurz auf die einzelnen Module für die Bilderzeugung und den Evolutionsprozess eingegangen. Nun werden der Bilderzeugungsteil und seine Module etwas genauer betrachtet.

Wort-Generierung:
Die Wort-Generierung erhält ein L-System als Eingabe. Dabei wird geprüft, ob das L-System vollständig ist und syntaktisch einwandfrei. Mit vollständig ist gemeint, dass die Rekursionstiefe, die Regeln und ein Axiom definiert sind. Mit syntaktisch einwandfrei ist gemeint, dass die Produktionen den Regeln der Syntax des Wort-Generierungsprozesses entsprechen. Der Prozess erkennt anhand der Grammatik um was für ein L-System es sich handelt und leitet das L-System entsprechend der Anzahl der Rekursionsschritte ab. Als Ergebnis entsteht ein einzelnes Wort, welches die Strukturvorlage für das Bild liefert. Diese Struktur ist abhängig von der Interpretation der einzelnen Zeichen des Wortes.

Modeller:
Die Aufgabe des Modellers ist es, aus einem generierten Wort eine 3D-Szene zu beschreiben. Dazu bekommt der Modeller als Eingabe das Wort aus dem vorherigen Schritt. Um die Strukturinformation zu interpretieren, benötigt der Modeller eine Interpretationsvorschrift. Sie schreibt vor, welches Symbol für welches Turtle-Kommando steht. In der Regel sind die Standardsymbole für die Turtle-Kommandos in der Interpretationsvorschrift enthalten. Die eigentliche Aufgabe des Modellers ist es, die Turtle-Kommandos, wie zum Beispiel F, in 3D-Primitive umzuwandeln und diese den Strukturinformation entsprechend anzuordnen. Als 3D-Primitiv wird üblicherweise der Zylinder verwendet. Einige Programme zu L-Systemen geben auch die Möglichkeit vor, andere Primitive zu verwenden, wie zum Beispiel einen Kubus. Durch die von Prusinkiewicz in [PL90] vorgenommene Erweiterung der Turtle-Interpretation ist es darüber hinaus auch möglich, komplexe Objekte mit Hilfe von Polygone zu modellieren. Polygone werden von vielen Renderern als 3D-Primitiv unterstützt, daher lassen sich auch Polygone, die im generierten Wort enthalten sind, problemlos in die 3D-Szene umzusetzen. Zusätzlich zu den Primitiven muss der Modeller auch eine Kamera einfügen und diese auf die 3D-Szene ausrichten sowie mindestens eine Lichtquelle, damit das 3D-Modell überhaupt sichtbar wird. Als Ergebnis erhält man eine 3D-Szene mit einem 3D-Modell samt Lichtquelle und Kamera.

Adapter:
Das 3D-Modell, das vom Modeller erstellt wird, ist nicht speziell auf einen Renderer zugeschnitten, bzw. liegt nicht in einem kompatiblen Format für einen speziellen Renderer vor. Diese Aufgabe muss der Adapter lösen. Der Adapter transformiert die allgemeine Beschreibung der 3D-Szene vom Modeller in eine spezielle Beschreibung für einen speziellen Renderer und speichert dies in dem Format des Renderers ab.

Renderer:
Aus der Beschreibung der 3D-Szene vom Adapter erzeugt der Renderer eine Grafik und speichert sie in einem Grafikformat ab. Der Renderer verwendet zur Erstellung der Szene die Strahlenverfolgungstechnik (Ray-Tracing). Weitere Information zum Renderer und zur Strahlenverfolgung folgen in Kapitel 4.5.

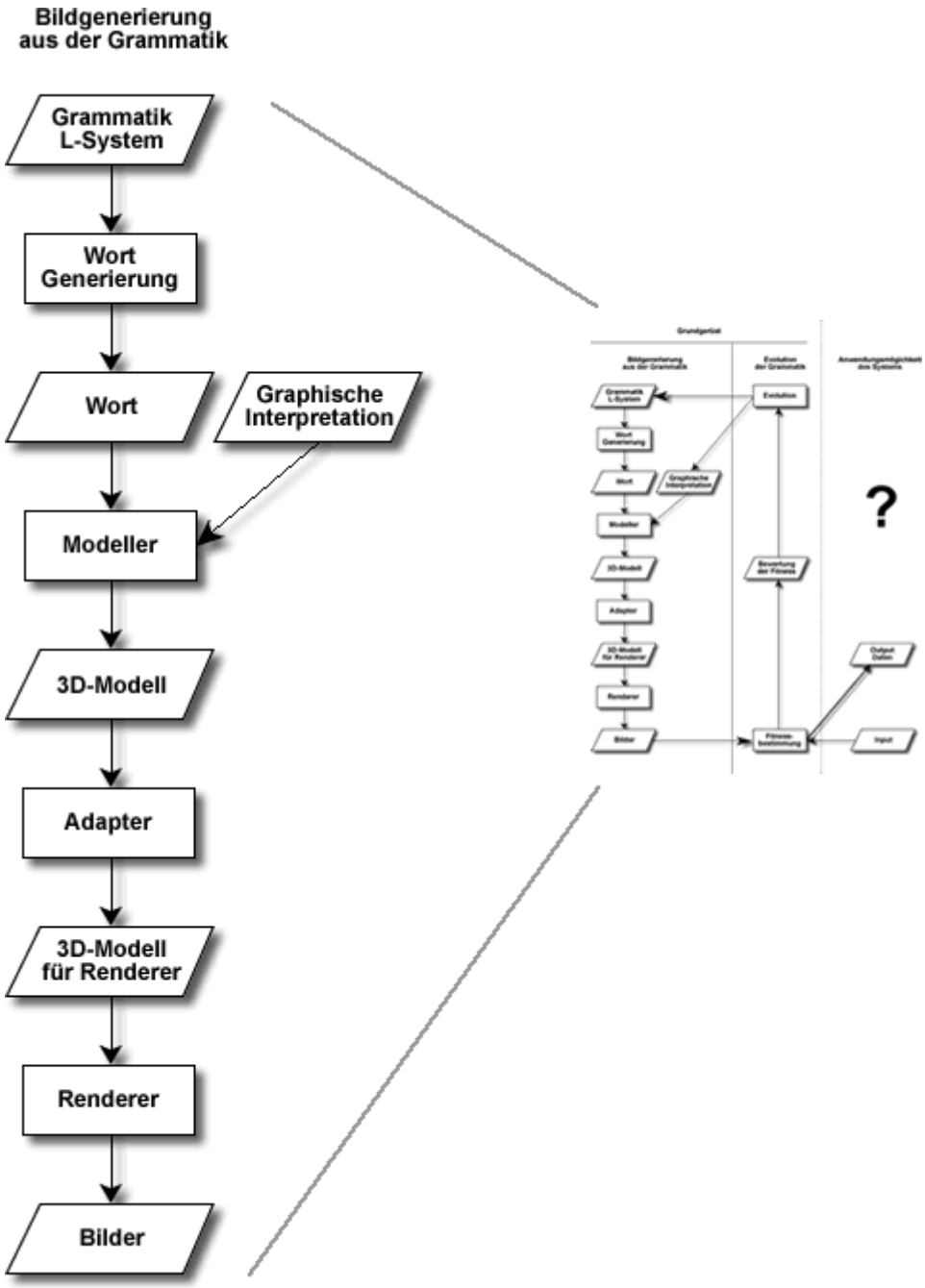


Abbildung 4.1: Module des Bilderzeugungsprozesses

4.5 Vereinfachungen und die entsprechende Modulübersicht

Im Rahmen einer Diplomarbeit können nicht alle Module selbst entwickelt werden. Es wird daher auf einige schon existierende Programme zurückgegriffen. In diesem Kapitel wird näher erläutert, um welche Programme es sich hierbei handelt und warum sie verwendet werden. Um einen ersten Überblick zu bekommen, werden der theoretische und der praktische Ablauf in Abbildung 4.2 einander gegenübergestellt. So wird deutlich, welches Programm welche Aufgabe aus dem theoretischen Teil übernehmen soll.

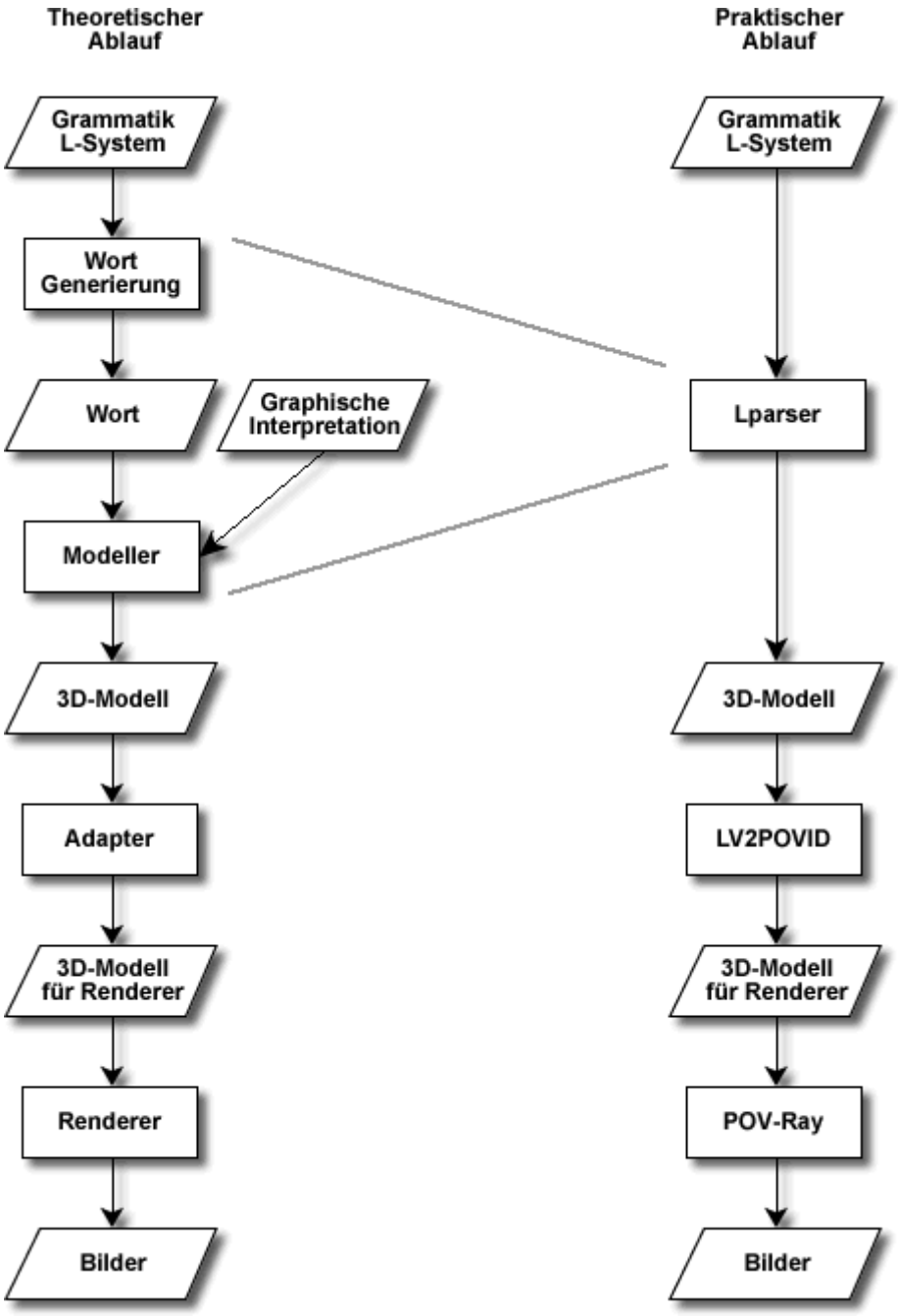


Abbildung 4.2: Der theoretische, der praktische Ablauf sowie die verwendeten Programme

Lparser und LV2POVID:
Wie schon im Kapitel 3.1 erläutert, ist Lparser ein Softwarepaket aus mehreren Komponenten. Für die Eingabe wird nur eine Text-Datei benötigt, in der ein L-System nach den syntaktischen Regeln vom Lparser enthalten ist. Des Weiteren wird aus dem Softwarepaket noch das Programm LV2POVID benötigt, um das 3D-Modell von Lparser in dass von POV-Ray zu überführen. Im nächsten Kapitel wird näher auf Lparser eingegangen. An dieser Stelle soll zunächst erklärt werden, warum Lparser ausgewählt wurde.
Um einen Prozess zu erstellen, der automatisch ablaufen soll, ist es notwendig, dass eine Schnittstelle existiert, wenn ein externes Programm eingebettet werden soll. Weiterhin ist es von Vorteil, wenn das Programm keine graphische Oberfläche enthält, denn es soll keine Interaktion mit einem Nutzer existieren. Daher bietet es sich an ein Programm zu verwenden, das über die Kommandozeile gestartet werden kann.

Anhand der Erläuterungen in Kapitel 3 lassen sich zwei Programme erkennen, die über die Konsole bedient werden können, eines davon ist Lparser und das andere RayTraced Evolution. RayTraced Evolution besitzt gegenüber Lparser einige Vorteile. So unterstützt es wesentlich mehr L-Systeme und kann durch das Treiber-Konzept theoretisch jeden beliebigen Renderer zur Bilderzeugung verwenden. Außerdem besitzt es eine mächtige, hochsprachenähnliche Sprache, mit der sehr komplexe Modelle erzeugt werden können. RayTraced Evolution hat einen gravierenden Nachteil, der das tiefer gehende Arbeiten mit dem Programm fast unmöglich macht. Es besitzt nur eine sehr knappe und nicht vollständige Dokumentation. Der komplette Aufbau einer Beschreibungs-Datei ist nicht erklärt. Das Gleiche gilt für viele Funktionen, die in der Abarbeitung eine wichtige Rolle spielen.
Der Vorteil von Lparser liegt in seiner weiten Verbreitung, denn dadurch bietet sich Möglichkeiten, schnell und umfassend Information zu dem Programm im Internet zu erhalten. Des Weiteren ist Lparser ein Open-Source Programm, das heißt der Quellcode kann herunter geladen und beliebig verändert werden, ohne dabei eine Lizenzbestimmung zu verletzen. Dies gilt aber nur für das Hauptprogramm Lparser selbst, andere Teile des Softwarepaketes sind nicht Open-Source, beispielsweise die Programme LViewer und LV2POVID. Lparser hat dafür einige andere Nachteile. So besitzt die Version 4 nur die Möglichkeit, PDB0L-Systeme zu interpretieren. Lediglich die Kommandos der Turtle-Interpretation sind parametrisiert. Darüber hinaus sind einige Teile von Lparser nicht fehlerfrei. So werden zum Beispiel von LViewer immer die gleichen Kamerapositionen für vollkommen unterschiedliche 3D-Modelle erzeugt und LV2POVID übernimmt diese Daten nicht richtig in die POV-Ray Datei. Aber der wichtigste Grund hinsichtlich der Entscheidung für Lparser ist, dass die Syntax der Beschreibungs-Datei einfach und intuitiv ist. Das macht das Verstehen von L-Systemen viel leichter und damit auch deren Gebrauch. Außerdem kann der Evolutionsprozess so leichter die L-Systeme verändern und selbst erstellte L-Systeme können sehr einfach in eine Datei gespeichert werden. Bei RayTraced Evolution müsste der Evolutionsprozess eine Beschreibungs-Datei mit mehreren Unterfunktionen erzeugen und die L-Systeme teilweise in ein hochsprachenähnliches Konstrukt umformen.

POV-Ray:
Im Bereich der Ray-Tracing Software gibt es nur sehr wenige Renderer, die frei verfügbar sind, und diese erzeugen selten so qualitative Szenen wie POV-Ray. Das liegt teilweise daran, dass POV-Ray auf eine sehr lange Entwicklungsgeschichte zurückschauen kann. Im Vergleich zu anderen Renderern, wie zum Beispiel NFF, der von dem Autor stammt, der auch RayTraced Evolution entwickelte, ist POV-Ray um viele Größenordnungen performanter. So benötigt der NFF Ray Tracer für eine Beispielszene, die mit RayTraced Evolution erstellt wurde, knapp zwei Minuten. POV-Ray benötigt für dieselbe Szene, die auch mit RayTraced Evolution erstellt wurde, keine zehn Sekunden. Weitere Vorteile von POV-Ray sind die hervorragenden Dokumentationen sowie die große Verbreitung. So erhält man in Newsgroups wie comp.graphics.rendering.raytracing innerhalb kürzester Zeit Antworten auf Fragen zu POV-Ray. Alternativ kann man auch über den Server von POV-Ray auf eine spezielle Newsgroup zugreifen und dort seine Fragen stellen. Zusammengefasst kann man sagen, dass POV-Ray einer der schnellsten freien Renderer mit einer sehr hohen Qualität bei der Bilderzeugung ist. Daher wird der Renderer POV-Ray gewählt.

4.6 Lparser

Die einzelnen Bestandteile von Lparser wurden schon in Kapitel 3.1 vorgestellt. An dieser Stelle wird zusätzlich dazu auf die Probleme mit dem Lparser eingegangen.
Zuvor wird jedoch eine Weiterentwicklung von Lparser näher erläutert.
Lparser in der Version 4 besitzt einige Nachteile, beispielsweise die geringe Unterstützung von verschiedenen L-Systemen oder die Abhängigkeit vom DOS4GW-Treiber. Weil Laurens Lapre den Quellcode von Lparser freigegeben hat, konnten zahlreiche Entwickler den Lparser erweitern. Eine dieser Personen ist Ken Kopp. Er hat im Jahre 2000 den Lparser auf die Version 5.1 gestellt. Seine Version von Lparser benötigt keinen DOS4GW-Treiber und unterstützt parametrisierte und kontextsensitive L-Systeme. Da diese Erweiterungen die wesentlichen Nachteile von Lparser aufheben, wird für den Bilderzeugungsprozess auf diese erweiterte Version zurückgegriffen.[4]
Lparser benötigt als Eingabe eine LS-Datei, in der das L-System definiert ist. Dabei interpretiert das Programm die erste auftretende Zeichenkette als die Rekursionstiefe, wobei Kommentare nicht ausgewertet werden. Somit muss die Rekursionstiefe nicht zwangsweise in der ersten Zeile stehen. Danach wird der Basiswinkel für die Turtle-Interpretation angegeben sowie der Anfangswert für die Stärke, also die Breite eines Zylinder-Primitives. Die vierte auftretende Zeichenkette wird als Axiom interpretiert. Alle weiteren Zeichenketten werden als Regeln des L-Systems verstanden.

Beispiel 4.1 für eine einfache LS-Datei:
Das Beispiel zeigt ein vollständiges L-System für den Lparser.

```
5           /* Rekursionstiefe
18          /* Basiswinkel
20          /* Anfangsstärke
P           /* Axiom

/* Regeln
P -> I+[P+R]--//[--L]I[++L]-[PR]++PR
I -> FS[//&&L][//^^L]FS
S -> SFS
L -> ['{+f-ff-f+|+f-ff-f}]
R -> [&&&C'/W////W////W////W////W]
C -> FF
W -> ['^F][{&&&&-f+f|-f+f}]
```

Kommentare in einer LS-Datei werden durch „/*“ kenntlich gemacht. Ansonsten sind die Bedeutungen der Zeichen identisch mit denen der Turtle-Interpretation. Die allgemeine Syntax für eine Regel ist:

```
<linker Kontext>  <  <Symbol>  >  <rechter Kontext>
: <bool'scher Ausdruck>  ->  <rechte Seite der Regel>
```

Beispiel 4.2 für eine Regel in einer LS-Datei:
Das Beispiel zeigt eine Produktion, die alle Teile einer Produktion unterstützt.

```
a < b(x) > c : x > 5 -> a(4) c(x) FFF [+a(x)]
```

Ein Unterschied zwischen Version 4 und 5 besteht darin, dass sich die Syntax für die LS-Datei leicht verändert hat. Diese Unterschiede werden in Tabelle 4.1 dargestellt.

[4] Wenn im Folgenden von der Version 4 von Lparser gesprochen wird, so ist das Original von Laurens Lapre gemeint und mit Version 5 die erweiterte Variante von Ken Kopp.

Bedeutung	Version 4	Version 5
Kommentar	#	/*
Regelzuweisung	=	->
Rotieren um die z-Achse, entgegen dem Uhrzeigersinn	<	\
Rotieren um die z-Achse, Uhrzeigersinn	>	/
Symbol für das Ende der LS-Datei	@	keins

Tabelle 4.1: Unterschiede zwischen Version 4 und 5

Außerdem können define-Anweisungen verwendet werden, wie man sie in C kennt.
Trotz einer immensen Weiterentwicklung von Lparser, der Code hat sich mehr als verdoppelt, sind weiterhin einige Schwächen vorhanden. Lparser prüft nicht, ob plausible Werte für Rekursionstiefe, Basiswinkel, Stärke und Axiom vorliegen. Außerdem können Zeichen in einer Produktion auftauchen, für die es keine Produktion zum Ableiten gibt. Es existiert auch keine Prüfung, ob die Anzahl der öffnenden und schließenden, eckigen und geschweiften Klammern die gleiche ist. Dies wäre jedoch im Hinblick auf die Evolution von Interesse, weil L-Systeme generiert werden können, die nicht der strengen Definition der Grammatik entsprechen. So kann eine komplette Produktion entfernt werden, ohne weitere Produktionen abzuändern und es können willkürlich, eckige Klammern gesetzt werden. Außerdem werden auch „nicht L-Systeme“ interpretiert, weil diese Prüfungen nicht stattfindet. Bezogen auf die Produktionsregeln sei noch darauf hingewiesen, dass anstelle eines Symbols auch „*“ als Wildcard stehen kann, wenn zum Beispiel bei einer kontextsensitiven Regel der linke Kontext nicht verwendet wird. Gleiches gilt auch für bool’sche Ausdrücke. Wenn die Regel auf jeden Fall angewandt werden soll, dann kann das Wildcardzeichen eingesetzt werden. Außerdem ist man nicht gezwungen, die Produktion vollständig zu beschreiben. Wenn es keine kontextsensitive Regel ist, dann kann der kontextsensitive Teil wegfallen. Das Gleiche gilt für den Bedingungsteil von L-Systemen.

Beispiel 4.3 für Regeln in LS-Dateien:
Im Folgenden werden einige Produktionen angegeben, die in einer LS-Datei auftreten könnten.

```
a               -> F + F + F + b(5)
b(x) : x > 4    -> c F f f + F a
* < c > F       -> [ - F - F a]
```

Ein Fehler in Lparser ist die Interpretation der Kommandozeilenargumente. Nicht alle Kommandozeilenargumente werden zuverlässig interpretiert. Das liegt daran, dass Lparser zur Auswertung der Argumente auf eine Funktion namens `Get_Comline_opt()` zurückgreift. Für diese Funktion wird vorher ein so genannter „Option String“ definiert, aus dem die gültigen Optionen abgeleitet werden. Um Lparser problemlos in den Bilderzeugungsprozess zu integrieren, wird das Programm erweitert, unter anderem auch die Anzahl der Kommandozeilenargumente. Als die neuen Argumente an den Option String angehängt wurden, hat Lparser beim Aufruf mit den Argumenten immer einen Fehler geworfen. Erst nachdem die Position der neuen Argumente im Option String auf die Mitte verlegt wurde, wurden keine Fehler mehr geworfen. Ungewöhnlich ist die im Programm verwendete Vektorschreibweise für die Punkte zur Erzeugung von Primitiven. So wird beispielsweise für die Anfangs- und Endpunkt eines Zylinders an erster Stelle die x-Koordinate angegeben, an zweiter die z-Koordinate und an dritter Stelle die y-Koordinate. Dies wird selbst bei POV-Beschreibungen für die 3D-Primitiven fortgesetzt, obwohl diese normalerweise die Reihenfolge (x, y, z) benötigen.

Normalerweise müsste im Anschluss nach dem Lparser das Programm LViewer ausgeführt werden. Denn durch das Programm LViewer wird eine Datei namens INFO.TXT erstellt, in der die Kameraeinstellungen gespeichert werden. Diese Datei benötigt wiederum LV2POVID für die POV-Ray Datei. Der LViewer hat aber einige Nachteile. So benötigt er den DOS4GW-Treiber und die Erstellung der INFO.TXT Datei geschieht nicht automatisch, sondern erst durch das Drücken der Taste F2. Außerdem sind die geschriebenen Daten in der INFO.TXT nicht korrekt, denn LViewer fügt immer wieder die gleichen Daten ein, selbst bei vollkommen unterschiedlichen Szenen. Daher wurde die Lparser Version 5 insoweit erweitert, dass bei Erstellung einer POV-Ray Datei Lparser selbst die INFO.TXT Datei erzeugt und eine sinnvolle Kameraposition berechnet.
Auch das dritte Programm im Softwarepaket von Lparser funktioniert nicht wie erwartet. Aufgabe des Programms LV2POVID ist es, die nicht POV-Ray gerechte Datei in eine POV-Ray gerechte Datei um zuwandeln. Außerdem liest das Programm die INFO.TXT Datei aus und setzt anhand der Daten die Kamera und die Lichtquelle. Es zeigt sich aber, dass an Stelle der Werte für die Kameraposition die Werte für die Kameraausrichtung eingesetzt werden und dass die Kameraausrichtung immer die Werte (0, 0, 0) zugewiesen bekommt. Außerdem wird mehrfach die Lichtquelle direkt im Körper des 3D-Modells platziert. Interessanterweise liest das Programm LV2POVID die INFO.TXT Datei richtig aus und verdreht nicht die Koordinaten, wie das bei Lparser der Fall ist. Das liegt vielleicht auch daran, dass die Daten in der INFO.TXT schon verdreht sind. Eine weitere Schwachstelle von LV2POVID ist, dass Dreiecke so beschrieben werden, wie bei POV-Ray in Version 2.X. Die aktuelle POV-Ray-Version 3.5 gibt zwar Warnungen aus, dass die Dreiecke nicht in der empfohlenen Syntax vorliegen, erzeugt aber die Dreiecke noch richtig. Außerdem gibt LV2POVID Daten an, die nicht notwendig sind, bzw. Daten, die zu einem falschen Ergebnis führen, wie zum Beispiel die Direction- oder Angle-Anweisung für die Kamera.

4.7 Persistence of Vision-Ray

1986 beginnt David K. Buck auf dem Amiga die Entwicklung eines Ray-Tracing Programms namens DKBTrace. 1989 wird der Name DKBTrace in POV-Ray (POV = Persistence Of Vision) geändert, weil mittlerweile sehr viele Menschen das Programm weiterentwickelten haben und Buck nicht als Projektleiter aufgefasst werden will. Damit ist POV-Ray [[Pov]] eines der ältesten Ray-Tracing Programme und es ist auch eines der bekanntesten. POV-Ray ist ein freier Renderer, dessen Quelldateien für jedermann zugänglich sind. Darüber hinaus gibt es zahlreiche Tools für POV-Ray. Eines der bekanntesten ist MORay, eine graphische Oberfläche zur Modellierung von 3D-Modellen.
POV-Ray basiert auf der Strahlenverfolgung. Das ist ein Prozess zur Simulierung der Lichtausbreitung in einer Szene. Zur Vereinfachung wird bei der Strahlenverfolgung nicht jeder Lichtstrahl von der Lichtquelle aus bis zum Auge verfolgt, sondern genau anders herum wird jeder Strahl eines Pixels auf dem Bildschirm so lange verfolgt, bis er eine Lichtquelle erreicht, bis die Energie des Strahls zu schwach ist oder bis der Strahl die Szene verlässt. Trifft der Strahl auf ein Objekt, wird die Beleuchtung an diesem Punkt berechnet und es werden zwei neue Strahlen erzeugt, einer der reflektiert wird und einer der gebrochen wird.

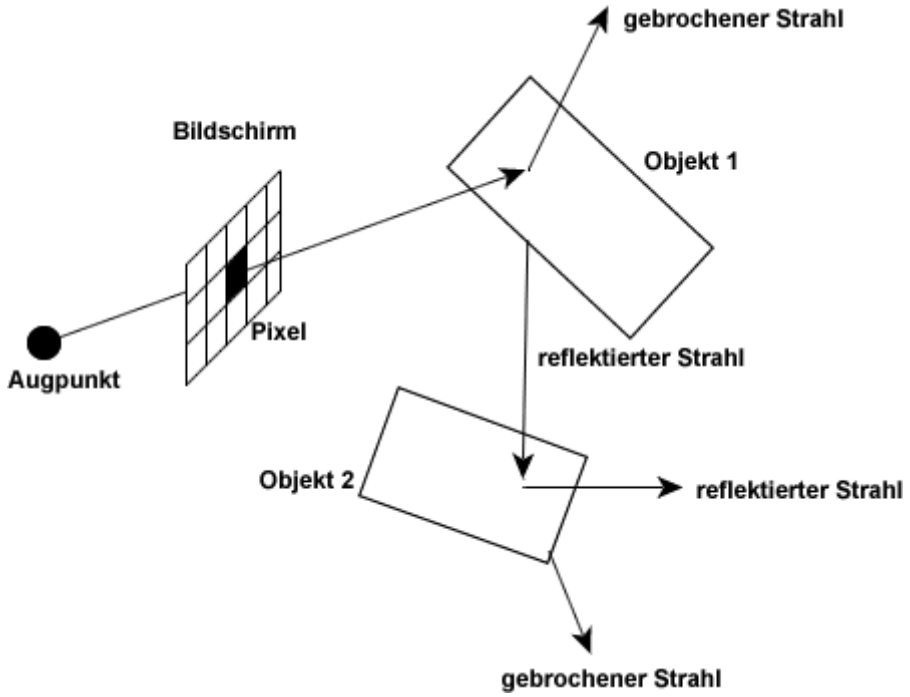


Abbildung 4.3: Strahlenverfolgung in einer Szene [ESK97]

Die Entwicklung einer 3D-Szene in POV-Ray läuft anders ab als bei Ray-Tracing Programmen wie 3D-Studio oder Lightwave. So kommt POV-Ray zwar mit einer graphischen Oberfläche, diese ist aber nicht zum graphischen Modellieren der 3D-Szene geeignet, sondern zum textbasierten Beschreiben der 3D-Szene. Somit ist die graphische Oberfläche ein Text-Editor, der auch zum Schreiben von Programmen genutzt werden kann, weil der POV-Ray Editor Syntax-Highlighting für verschiedene Programmiersprachen enthält.
Die Beschreibungssprache von POV-Ray ist an C/C++ angelehnt und ebenfalls case-sensitiv. Als Kommentarzeichen kann // für einen Zeilenkommentar oder /* und */ als Kommentar über mehrere Zeilen genutzt werden, sie sind also identisch mit den Kommentarmöglichkeiten in C++.
Ebenfalls identisch mit C/C++ ist die Bedeutung der #include-Anweisung. Somit können Beschreibungsteile ausgegliedert werden oder als allgemeine Modelle für andere 3D-Szenen genutzt werden. Die #declare-Anweisung wird zum Beispiel benutzt, um Konstanten zu definieren. Die Anweisung kann aber auch benutzt werden, um ein neues 3D-Objekt zu definieren. Elemente einer 3D-Szene werden stattdessen mit ihrem Namen aufgerufen und in geschweiften Klammern näher spezifiziert. Die Reihenfolge der 3D-Elemente, #include-Anweisungen und #declare-Anweisung spielt an sich keine Rolle. Wenn aber auf den Inhalt einer Include-Datei zugegriffen wird, dann muss diese auch vorher deklariert sein. Gleiches gilt für Konstanten oder neue 3D-Objekte, die mit der #declare-Anweisung definiert wurden, auch diese müssen vor der Nutzung bekannt sein. Um eine bessere und nicht fehleranfällige Struktur der Beschreibung zu erlangen, sollten am Anfang alle #include-Anweisungen stehen, gefolgt von den #declare-Anweisungen. Weiterhin hat POV-Ray in seiner Beschreibungssprache auch Direktiven wie IF und WHILE.
Eine sinnvolle 3D-Szene besteht mindestens aus einer Kamera, einer Lichtquelle und einem Objekt. Wie eine solche Szene aussehen könnte, wird an folgendem Beispiel deutlich.

Beispiel 4.4 für eine 3D-Szene:
Es wird die Beschreibung für eine einfache 3D-Szene unter POV-Ray angegeben.

```
#include "colors.inc"
#include "glass_old.inc"
#include "metals.inc"

camera {
        location <1.5, 10, -10>         // Kameraposition
        look_at  <1.5, 1, 2>            // Kameraausrichtung
}
light_source {
        <500 , 1000, -50>               // Position x, y, z
        color White                     // Farbe
}
plane {                                 // Ebene
        <0, 1, 0>,                      // Oberflächennormale
        -1                              // Position zur y-Achse
        pigment {                       // Muster Schachbrett
                checker color White     // Schwarz/Weiß
                        color Black
                scale 4                 // Skalierung
        }
}
sphere {
        <0, 1.5, 2>,                    // Mittelpunkt
        3                               // Radius
        texture {                       // Textur
                T_Chrome_5A             // Chrome
                pigment {               // Blau
                        color Blue
                }
        }
}
sphere {
        <3, 1.5, 2>, 3
        texture {
                T_Glass1                // Glastextur
        }
}
```

POV-Ray kennt als 3D-Primitive die Sphäre (Sphere), das Rechteck (Box), den Zylinder (Cylinder), den Torus, den Kegel (Cone) und die Ebene (Plane). Mit Hilfe von CSG (Constructive Solid Geometry) lassen sich aus den 3D-Primitiven neue komplexe 3D-Objekte erstellen, wie zum Beispiel eine Kette. Dazu werden Objekte in CSG-Operationen eingebettet, wobei die CSG-Operationen die Mengen-Operationen abbilden. Als Operatoren existieren Vereinigung (Union), Schnittmenge (Intersection) und Differenzmenge (Difference). Zusätzlich gibt es noch den Operator Merge zum Verbinden von Objekten, er kommt der Vereinigung gleich. Außerdem kennt POV-Ray auch Polygonen und auf Spline basierenden Formen sowie verschiedene Arten von Lichtquellen, wie zum Beispiel Pointlight, Spotlight, Ambient Light und andere.
Für weitere Informationen sei auf die Dokumentation von POV-Ray verwiesen, auf http://www.povray.org/documentation/.
[ESK96] [ESK97]

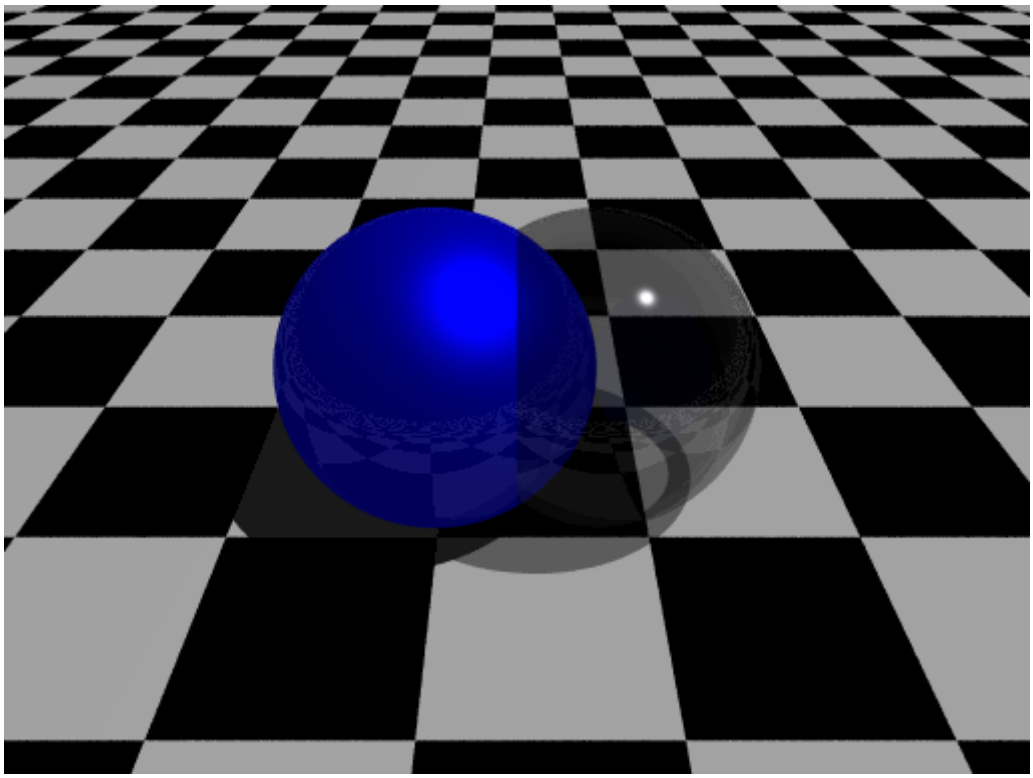


Abbildung 4.4: Die gerenderte Szene

4.8 Schnittstelle der Fassade

Mit der Fassade werden zwei Wege angeboten, diverse Einstellungen festzulegen. Der erste Weg sind die Kommandozeilenargumente. In Tabelle 4.2 sind alle Kommandozeilenargumente und deren Auswirkungen aufgelistet.

Kommandozeilenargument	Auswirkung
-?	Ruft einen Bildschirm auf mit Erklärungen zu den Kommandozeilenargumenten.
-H[Zahl]	Gibt die Höhe der Grafik in Pixel an.
-W[Zahl]	Gibt die Breite der Grafik in Pixel an.
-T	Das Ausgabeformat ist eine Targa-Grafikdatei.
-A	Aktiviert die Anti-Aliasing Funktion.
-C	Konvertiert eine Datei in die Version 5.
-c	Konvertiert eine Datei in die Version 5 und beendet danach den Prozess.
-E	Beendet POV-Ray nachdem der Prozess das Bild erzeugt hat.
-e	Beendet POV-Ray ohne das Starten einer Bilderzeugung.
-F	Dieses Argument gibt an, dass alle Dateien aus den Zwischenschritten erhalten bleiben sollen.

Tabelle 4.2: Kommandozeilenargumente der Fassade

Die Aufrufskonventionen der Kommandozeile sind:

```
lprocess [Optionen] [zu konvertierende LS-Datei] [LS-Datei]
```

Die Optionen entsprechen den Kommandozeilenargumenten. Diese müssen nicht einzeln angegeben werden, sie können auch kombiniert werden, wie zum Beispiel:

```
lprocess -ACEW640H480 bekerpl bekerpl5
```

Das Aufrufen der Fassade würde bei diesen Argumenten die Anti-Aliasing Funktion aktivieren, die Datei bekerpl in die Version 5 konvertieren und in bekerpl5 abspeichern, POV-Ray nach Beendigung des Prozesses ebenfalls beenden und die Größe der 3D-Grafik auf 640*480 Bildpunkte festlegen. Wird die Konvertierungsfunktion aktiviert, dann können auch zwei Dateinamen angegeben werden. In solch einem Fall speichert der Prozess die Konvertierung im zweiten angegebenen Dateinamen ab. Wird nur ein Dateiname angegeben, dann wird der alte Inhalt durch den neuen ausgetauscht.
Neben der Kommandozeile gibt es noch eine Konfigurationsdatei (LPROCESS.CFG), in der einige Werte für diverse Einstellungen stehen müssen. Ohne eine vollständige Konfigurationsdatei startet die Fassade nicht den Bilderzeugungsprozess. In Tabelle 4.3 sind alle Einstellungen, deren Werte und ihre Bedeutung aufgelistet.

Einstellungen	Werte	Bedeutung
Image	„bmp“ oder „targa“	Speichert die Grafik als Bitmap (bmp) oder als Targa.
Height	Positive ganze Zahl	Gibt die Höhe der Grafik in Pixel an.
Width	Positive ganze Zahl	Gibt die Breite der Grafik in Pixel an.
AA	„on” oder „off”	Gibt an, ob die Anti-Aliasing Funktion aktiviert werden soll.
POV	Pfad	Diese Einstellung muss den Pfad für POV-Ray beinhalten.
QPOV	Pfad	Diese Einstellung muss den Pfad für QuietPOV enthalten.
Files	„on” oder „off”	Durch Setzen der Einstellung, werden alle Dateien aus den Zwischenschritten erhalten.
CamX	Reelle Zahl	Diese Zahl wird benutzt, um mit der x-Koordinate der Kameraposition multipliziert zu werden.
CamY	Reelle Zahl	Diese Zahl wird benutzt, um mit der y-Koordinate der Kameraposition multipliziert zu werden.
CamZ	Reelle Zahl	Diese Zahl wird benutzt, um mit der z-Koordinate der Kameraposition multipliziert zu werden.
LightX	Reelle Zahl	Mit dieser Zahl wird die x-Koordinate der Kamera multipliziert, zur relativen Positionierung der Lichtquelle.
LightY	Reelle Zahl	Mit dieser Zahl wird die y-Koordinate der Kamera multipliziert, zur relativen Positionierung der Lichtquelle.
LightZ	Reelle Zahl	Mit dieser Zahl wird die z-Koordinate der Kamera multipliziert, zur relativen Positionierung der Lichtquelle.
//	keinen	Kommentarzeichen

Tabelle 4.3: Alle Werte der LPROCESS.CFG

Wobei jede Einstellung, außer den Kommentaren, in der Form

```
Einstellung=Wert
```

erfolgen muss. Des Weiteren gilt, dass alle Werte von der Kommandozeile eine höhere Priorität haben als in der CFG-Datei. Wenn zum Beispiel beim Kommandozeilenaufruf die Anti-Aliasing Funktion angegeben wird und in der CFG-Datei steht ein „off“, dann wird das Anti-Aliasing trotzdem für die Bilderzeugung mit benutzt. Die Datei LPROCESS.CFG ist eine einfache Textdatei.

4.9 Ablauf der Fassade

Bevor auf die Implementierung der Fassade eingegangen wird, soll der Ablauf der Verarbeitung in der Fassade geklärt werden. Dazu ist in Abbildung 4.5 das Ablaufdiagramm dargestellt, welches nachfolgend besprochen wird. Eine Raute symbolisiert dabei die Ausführung einer Aktion und deren Auswertung. Ein Rechteck steht für das Ausführen einer Aktion, in der keine Auswertung stattfindet.

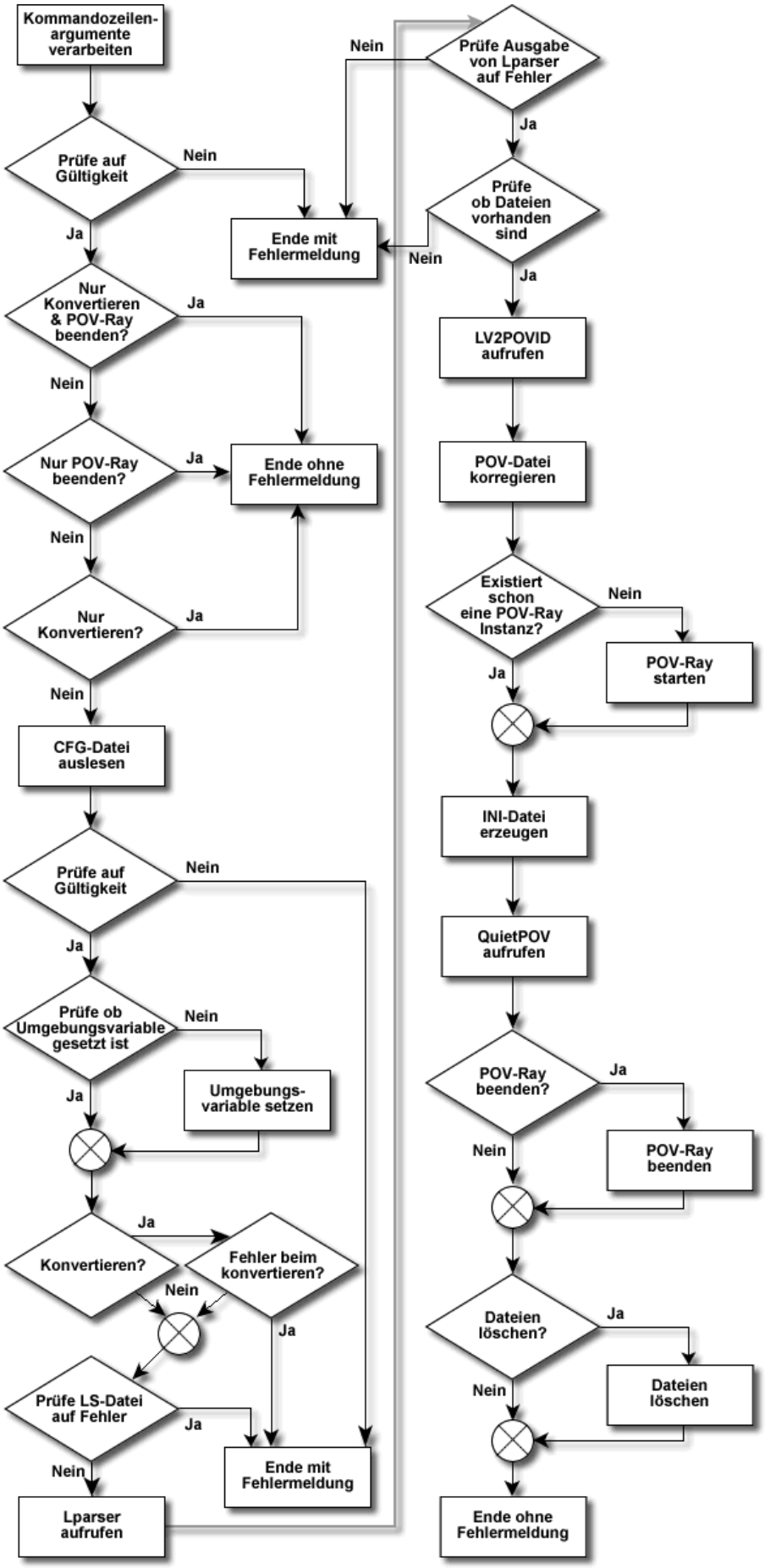


Abbildung 4.5: Ablaufdiagramm der Fassade

Nach dem Start der Fassade werden die Kommandozeilenargumente verarbeitet, d. h. es wird geprüft, ob die Kommandozeilenargumente und deren Kombinationen gültig und vollständig sind. So wird beispielsweise die Kombination „-e -E“ nicht zugelassen, weil verlangt wird, dass POV-Ray sofort und nach der Verarbeitung beendet wird. Außerdem wird geprüft, ob Dateinamen vorliegen und ob nach den Dateinamen noch weitere Eingaben vorhanden sind. Wurde bei der Prüfung ein Fehler festgestellt, wird das Programm mit einer Fehlermeldung beendet.
Bevor der Prozess zur Bilderzeugung weiter abgearbeitet wird, wird zuvor geprüft, ob die vollständige Abarbeitung für die definierte Zielsetzung notwendig ist. Wird beispielsweise als Kommandozeilenargument angegeben, dass nur eine Datei konvertiert oder nur POV-Ray beendet werden soll, dann wird die Aktion auf der Stelle ausgeführt und das Programm ohne Fehlermeldung und ohne ein Bild zu erzeugen beendet.
Die zweite Schnittstelle zur Fassade bildet der Inhalt der CFG-Datei. Dieser Inhalt wird komplett ausgelesen und zwischengespeichert. Danach werden alle Variablen, die noch nicht durch ein Kommandozeilenargument gesetzt wurden, durch den Inhalt der CFG-Datei gesetzt. Stellt das Programm bei der Wertzuweisung an die Variablen fest, das kein Wert oder kein gültiger Wert vorliegt, wird das Programm mit einer Fehlermeldung beendet.
Der letzte Vorbereitungsschritt besteht darin zu prüfen, ob für das Verzeichnis von QuietPOV eine Umgebungsvariable gesetzt wurde. Sollte dies nicht der Fall sein, dann wird für die Verarbeitungszeit die Umgebungsvariable gesetzt.
Als ersten Verarbeitungsschritt konvertiert die Fassade eine LS-Datei von Version 4 in die Version 5, falls dies angegeben wurde. Sollte während der Verarbeitung ein Fehler auftauchen, wird die Fassade mit einer Fehlermeldung beendet.
Bevor Lparser mit einer LS-Datei gestartet wird, wird die LS-Datei nach einem gültigen Inhalt geprüft, das heißt es wird überprüft, ob eine Rekursionstiefe, ein Startwinkel, eine Startdicke und ein Axiom angegeben wurden. Diese Prüfung entspricht mehr einer Plausibilitätsprüfung. Außerdem wird geprüft, ob runde Klammern im linken oder rechten Kontext bzw. im Bedingungsteil vorliegen. Falls kein plausibeler Wert vorliegt oder runde Klammern vorliegen, beendet das Programm mit einer Fehlermeldung.
Lparser bekommt als Eingabe die LS-Datei und einige Kommandozeilenargumente. So soll das Programm eine Beschreibungsdatei für POV-Ray erzeugen und diese als INC-Datei speichern. Außerdem werden Lparser über die Kommandozeile die Faktoren übergeben, mit denen die Kameraposition multipliziert wird. Die Kameraposition wird dabei anhand der Ausdehnung der 3D-Szene berechnet. Ist die 3D-Szene eine echte 3D-Szene, das heißt die Ausdehnungen des Modells bezieht sich auf alle drei Achsen, dann werden die Kamerakoordinaten berechnet, indem eine Ausdehnung der jeweiligen Achse mit dem übergebenen Kommandozeilenwert multipliziert wird. Ziel ist es, die Kamera relativ zur Modellausdehnung zu positionieren. In Abbildung 4.6 a) wird dies verdeutlicht. Sollte keine echte 3D-Szene vorliegen, also eine Achsenausdehnung sowohl in negativer als auch positiver Richtung nahezu null sein, dann wird der Kamera von den Koordinaten, die eine Ausdehnung besitzen, deren Mittelwert zugewiesen. Für die Koordinate, die eine Ausdehnung nahezu null besitzt, wird ein neuer Wert berechnet. Dazu wird der Mittelwert der anderen beiden Koordinaten addiert und mit dem zur Achse gehörigen Wert von der Kommandozeile multipliziert. Ziel ist es, die Kamera auf die Mitte des 2D-Modells auszurichten und mit der Koordinatenachse, die nahezu null war, die Entfernung der Kamera zu regulieren. In Abbildung 4.6 b) wird eine x-Achse gezeigt, die keine Ausdehnung aufweist.

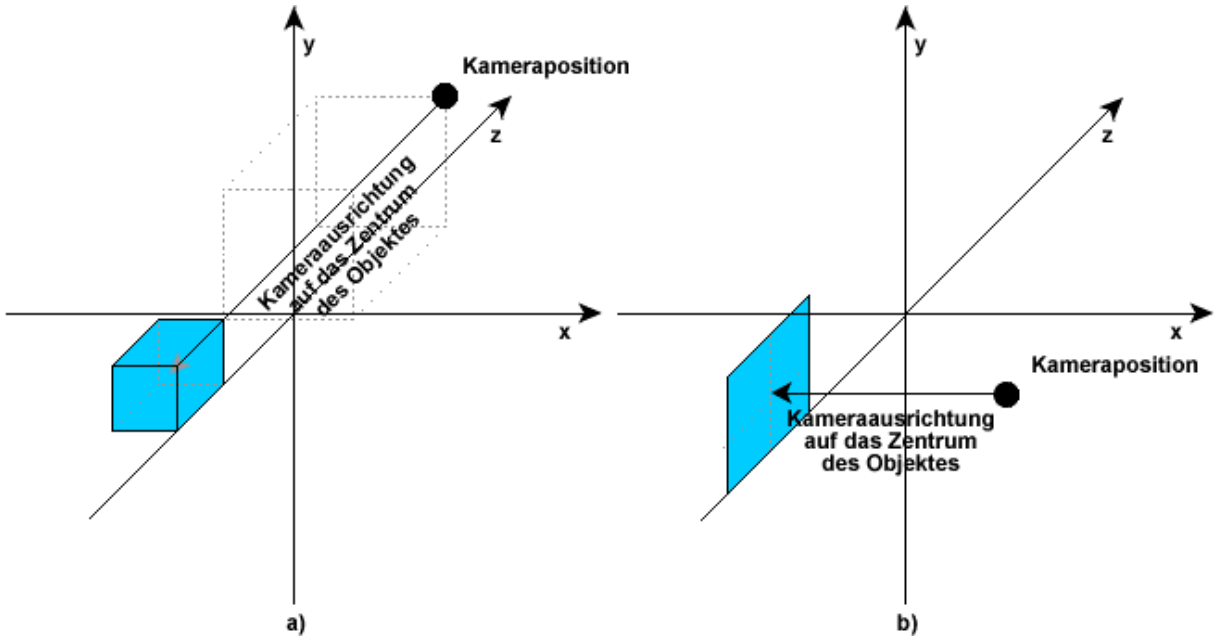


Abbildung 4.6: Beispielhafte Kameraposition für a) 3D-Szenen und b) 2D-Szenen

Nach Beendigung von Lparser wird anhand der Ausgabe von der Fassade geprüft, ob ein Fehler in einer Regel vorlag. Zusammen mit der Nummer der Regel wird dann eine Fehlermeldung ausgegeben.
Bevor LV2POVID gestartet wird muss geprüft werden, ob alle notwendigen Dateien vorhanden sind, das heißt ob die INFO.TXT, die LV2POVID.CFG und die OUTPUT.INC gefunden wurden. Die INFO.TXT enthält die Daten für die Kameraposition, wobei es kein Fehler ist, dass die Daten in der Datei doppelt vorhanden sind. LV2POVID erkennt die INFO.TXT nicht an, wenn die Daten nicht doppelt vorhanden sind. In der LV2POVID.CFG steht der Pfad für das Programm Fractint. Zwar wird das Programm Fractint nicht verwendet, trotzdem muss die Datei mit irgendeinem Pfad vorhanden sein, ansonsten startet LV2POVID nicht. Auch die von Lparser erzeugt Beschreibungs-Datei OUTPUT.INC muss vorhanden sein, denn wenn eine der drei Dateien nicht vorliegt, wird das Programm mit einer Fehlermeldung beendet.
Nachdem LV2POVID die OUTPUT.INC in eine richtige POV-Ray Datei umgewandelt hat, wird die von LV2POVID erzeugte POV-Ray Datei korrigiert. Die OUTPUT.INC heißt zu diesem Zeitpunkt LPAR2POV.INC. Die zu korrigierende Datei heißt LPAR2POV.POV. Sie beinhaltet die Daten für die Kamera und für die Lichtquelle. Da LV2POVID trotz der INFO.TXT die Daten nicht richtig in die POV-Datei einträgt, muss dies korrigiert werden. Daher wird in diesem Schritt die richtige Kameraposition nachgetragen und die Position der Lichtquelle berechnet und eingetragen. Die Lichtquelle wird relativ zur Kameraposition platziert. Ausgehend von den Koordinaten der Kamera werden dazu die Koordinaten mit den definierten Werten aus der CFG-Datei multipliziert.
Bevor das Bild gerendert wird, prüft die Fassade, ob schon eine Instanz von POV-Ray existiert. Falls das nicht so ist, wird POV-Ray gestartet.
Nun werden alle für POV-Ray relevanten Daten in eine INI-Datei gespeichert, darunter auch der Name der POV-Ray Datei und der späteren Bilddatei. Für die Ausgabedatei wird der Name der LS-Datei gewählt.
Im Anschluss wird das Programm QuietPOV mit der INI-Datei aufgerufen. QuietPOV ist ein Kommandozeilenprogramm mit dem es möglich ist, über die Kommandozeile Befehle an POV-Ray zu senden. Dazu nutzt das Programm die GUI-Extension-Schnittstelle von POV-Ray. Durch den Einsatz von QuietPOV wird das ständige Laden der graphischen Oberfläche und das damit verbundene Auftauchen des Splash-Screens unterbunden. Der Splash-Screen

darf aus lizenzrechtlichen Gründen nicht deaktiviert werden. Wenn POV-Ray nicht privat benutzt wird, muss deutlich kenntlich sein, dass POV-Ray benutzt wird. Dies geschieht durch den Splash-Screen.
Nachdem das Bild erzeugt ist wird geprüft, ob POV-Ray an dieser Stelle beendet werden soll. Zuletzt wird anhand des Kommandozeilenargumentes „-F“, entschieden, ob die Dateien, die zwischenzeitig erzeugt wurden, gelöscht werden sollen oder nicht.

4.10 Implementierung der Fassade

Bevor die eigentliche Fassade implementiert wird, muss Lparser auf die neuen Bedürfnisse, auf das Berechnen der Kameraposition und auf das Erzeugen der INFO.TXT angepasst werden. Wie bereits erwähnt, basiert Lparser Version 5.1 auf der Version 4. Version 4 wurde noch im reinen C geschrieben. Die Erweiterungen wurden von Ken Kopp mit C++ entwickelt. Auch die Anpassungen an Version 5.1, die für die vorliegende Arbeit notwendigen wurden, sind in Visual C++ 6.0 durchgeführt.
Um bei Lparser die maximale Ausdehnung der 3D-Szene zu erhalten, werden globale Variable definiert, die prüfen, ob die Koordinaten des Objektes größer sind, als die bisher gespeicherten, bevor ein Objekt in die POV-Ray Datei geschrieben wird. Sind die Koordinaten des aktuellen Objektes größer, sowohl in negativer als auch in positiver Richtung, dann werden diese in den Variablen gespeichert. Die Objekte werden in den Funktionen `Define_form()` und `Save_object()` abgefangen und geprüft.
Zusätzlich wird die Anzahl der Kommandozeilenargumente erhöht bzw. um `-x[num]`, `-y[num]` und `-z[num]` erweitert, damit für die Kameraposition die Faktoren vorliegen, mit denen sie multipliziert werden sollen. In der `main()`-Funktion werden alle Kommandozeilenargumente im Hinblick darauf geprüft, ob sie gesetzt wurden oder nicht. Dazu wird auf eine Hilfsfunktion namens `Get_comline_opt()` zurückgegriffen. Diese Funktion benötigt vor ihrem Aufruf die Zuweisung eines „Option String“, in dem alle Symbole enthalten sind, die als Argumente von der Kommandozeile übergeben werden können. Beim Aufruf von `Get_comline_opt()` muss das Symbol angegeben werden, nach dem gesucht wird, sowie eine bool’sche Variable die angibt, ob die Option vorhanden ist. Zusätzlich dazu wird ein char-Array angegeben, in dem der Wert des Kommandozeilenargumentes enthalten ist, falls für dieses Kommandozeilenargument ein Wert vorliegt.
Am Ende der `main()`-Funktion wird die Funktion `createINFOTXT()` aufgerufen, die dann die INFO.TXT Datei erzeugt und die Kameraposition berechnet. Zur Berechnung der Kameraposition wurden zwei Hilfsfunktionen entwickelt. Die eine heißt `dAbs()` und bildet die Betragsfunktion für double-Werte ab. Die zweite Funktion heißt `percOfAxisValues()`. Diese Funktion berechnet einen bestimmten Prozentwert von der größten Ausdehnung einer bestimmten Achse, die beim Aufruf angegeben wird.

Zusammenfassung:
Modifiziert wurden die Funktionen `main()`, `Define_form()` und `Save_object()`. Die globalen double-Variablen `maxX`, `minX`, `maxY`, `minY`, `maxZ`, `minZ`, `x`, `y`, `z` wurden hinzugefügt außerdem folgende Funktionen:

```
double dAbs(double number);
double percOfAxisValues(double perc, char type) ;
void   createINFOTXT(void);
```

Um, wie im Quelltext von Lparser, das umständliche Suchen von Funktionen zu unterbinden, wurden Funktionen gruppiert und in eigene C-Dateien ausgelagert. Somit ergibt sich eine Modularisierung gemäß Tabelle 4.4.

Modulname	Kurzbeschreibung
Lprocess.c	Enthält lediglich die `main()`-Funktion.
globals.h	Enthält wichtige Präprozessor-Anweisungen, globale Variablen und Strukturen, die für alle Module zur Verfügung stehen.
cfg_file.c / cfg_file.h	Alle Funktion, die zum Auslesen und Interpretieren der CFG-Datei benötigt werden, sind in dem Modul enthalten.
com_line.c / com_line.h	Enthält Funktionen zur Verarbeitung und Interpretation der Kommandozeilenargumente.
lparser.c / lparser.h	Das Modul enthält alle Funktionen, die mit dem Lparser-Paket interagieren.
misc.c / misc.h	Die einzelnen Funktionen, die keinem speziellen Modul zugewiesen werden können, finden sich hier wieder.
pov.c / pov.h	Alle Funktionen, die mit POV-Ray oder QuietPOV kommunizieren, sind in diesem Modul enthalten.

Tabelle 4.4: Module der Fassade

Die Datei Lprocess.c enthält lediglich die `main()`-Funktion. Diese entspricht im Groben dem Ablaufdiagramm aus Abbildung 4.6. Lediglich die Prüfung und das Starten einer POV-Ray Instanz wird von der Fassade vor dem Ausführen von Lparser getätigt. Dies ist nötig, damit POV-Ray ausreichend Zeit bekommt, die GUI-Extension-Schnittstelle zur Verfügung zustellen und Befehle über die Schnittstelle von QuietPOV entgegenzunehmen. Falls bei der Verarbeitung ein Fehler auftritt, gibt die Fassade den Wert -1 an das Betriebssystem zurück. Die globals.h enthält zwei wichtige Strukturen, die `InfoStruct` und die `CFGStruct`. In der `InfoStruct` sind alle wichtigen Informationen für den Verlauf der Bilderzeugung enthalten, die während der Verarbeitung benötigt werden. Der Inhalt der Variablen ergibt sich aus den Werten der Kommandozeile und aus den Optionen der CFG-Datei. So sind zum Beispiel Variablen in der `InfoStruct,` die die Höhe und Breite der Grafik in Pixel speichern oder Variablen, die den Pfad für POV-Ray und QuietPOV enthalten. Besonders hervorzuheben ist die Variable `state`. Diese 8-Bit Variable speichert in jedem Bit eine bestimmte Einstellung, beispielsweise ob als Ausgabeformat das Targa- oder das Bitmapformat verwendet werden soll. Daher kann die Variable mit Bitmaskierungen abgefragt und gesetzt werden. Die einzelnen Bedeutungen der Bits können aus der Abbildung 4.7 entnommen werden. Die Struktur `CFGStruct` wird als Zwischenspeicher zum Einlesen der CFG-Datei benutzt. Sie besteht aus zwei char-Arrays, die als `left` und `right` bezeichnet werden. In `left` wird die linke Optionsseite und damit auch der Bezeichner einer Option gespeichert, `right` speichert entsprechend die rechte Optionsseite und damit den Wert eines Bezeichners. Der restliche Inhalt der globals.h, besteht aus diversen Präprozessor-Anweisungen. So werden die Bitmasken für die einzelnen Werte von `state` definiert, ebenso deren nicht erlaubte Kombinationen. Außerdem werden diverse Maximumwerte definiert, wie zum Beispiel die maximale Länge einer Zeile in einer LS-Datei oder in der CFG-Datei. Eine wichtige `define`-Anweisung verbirgt sich hinter `POV_APP`. Der String enthält den Namen von POV-Ray, der die Adresse für Windows-Nachrichten darstellt. Windows-Programme erhalten über Nachrichten Informationen vom Betriebssystem, beispielsweise dafür, dass eine Taste für das Programm gedrückt wurde. So enthält jedes Windows-Programm eine Nachrichtenschleife, in der die jeweilige Nachricht ausgewertet wird. Tiefer-

gehende Informationen zur Windows-Programmierung sind in [Ri97] [DS98] [To99] [LA98] zu finden. Um eine POV-Ray Instanz zu finden oder zu beenden, benötigen die API-Funktionen von Windows den Applikationsnamen von POV-Ray. Dieser ist aber nicht identisch mit der Executable-Datei oder mit der Beschreibung der Datei. Um solch einen Applikationsnamen zu erhalten, kann man mit Tools die Nachrichtenverarbeitung von Windows beobachten. Eines dieser Tools heißt WinSight. Es stammt von Borland und ist in einigen Borland-Produkten, beispielsweise in Delphi enthalten. Der aktuelle Applikationsname von POV-Ray in der Version 3.5 lautet „Pov35MainWinClass“. Wenn eine neue Version erscheint, kann sich dieser Name ändern.

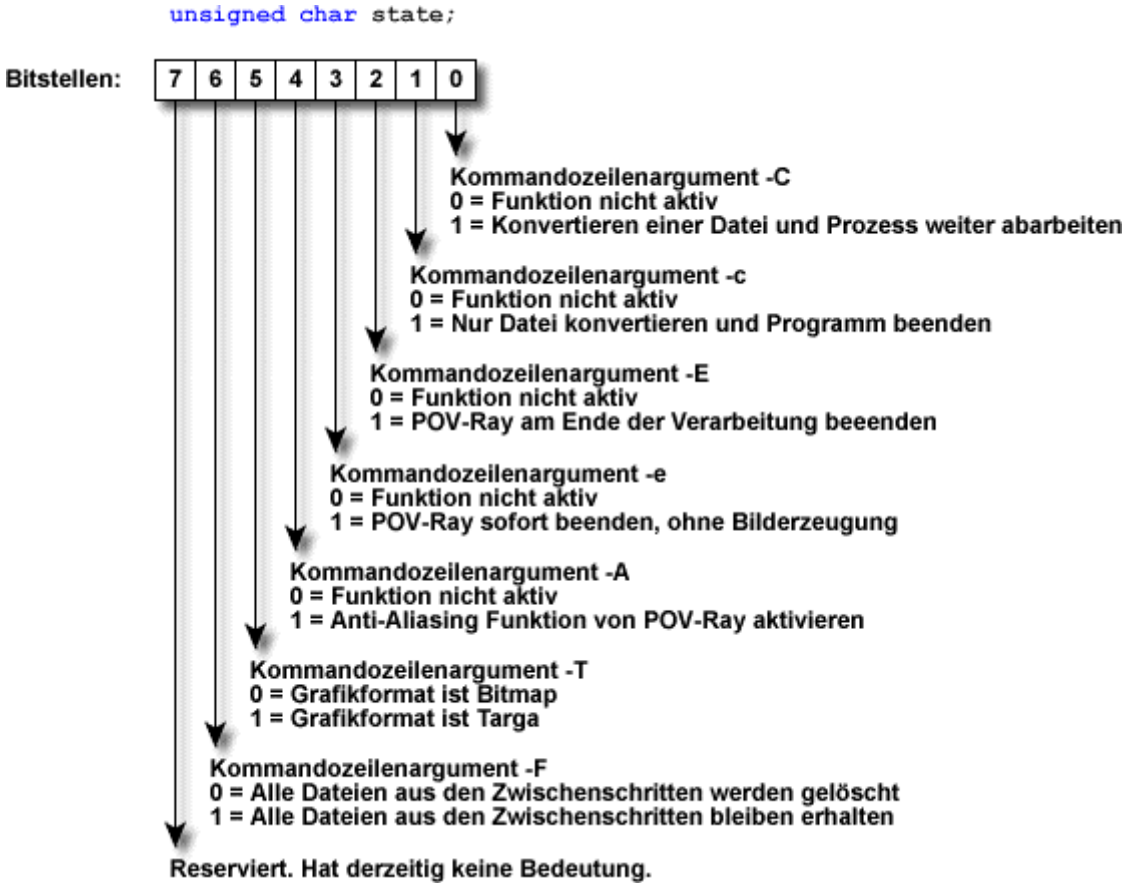


Abbildung 4.7: Bitbelegung der Variable `state`

Das Modul cfg_file.c enthält drei Funktionen, deren Aufgabe es ist, die Kommandozeilenargumente zu verarbeiten und zu interpretieren. Dabei wird die meiste Arbeit in der Funktion `checkingCFGFile()` durchgeführt. Diese Funktion ist auch die einzige Funktion, die nicht der Faustregel folgt, wenn eine Funktion mehr als zwei Bildschirmgrößen groß ist, so lagere Teile dieser Funktion aus. Auf eine Ausgliederung wurde in diesem Fall verzichtet, weil dies aufgrund der starken Verzahnung des Quelltextes nicht sinnvoll erschien. Alle anderen Funktionen orientieren sich an dieser Regel, weil dadurch die einzelnen Teile besser gewartet und getestet werden können. Außerdem erhöht die Ausgliederung auch die Übersichtlichkeit des Quelltextes [JAH01]. Die Funktion `checkingCFGFile()` versucht zunächst die Datei LPROCESS.CFG zu öffnen. Wenn der Versuch fehlschlägt, dann wird die `InfoStruct` mit Standardwerten gefüllt und die Funktion `createDefaultCFGFile()` aufgerufen, die eine CFG-Datei mit Standardwerten erzeugt. Danach wird in einer Schleife die Datei ausgelesen und gefundene Optionen in einer Variablen von der Struktur `CFGStruct` gespeichert. Im Anschluss werden alle Variablen aus der `InfoStruct` belegt,

die noch keinen gültigen Wert aufweisen. Zum Suchen eines Wertes aus der ausgelesenen Datei gibt es eine Hilfsfunktion namens `getValueFromList()`. Die Funktion bekommt einen Zeiger auf das Array aus Variablen der Struktur `CFGStruct` übergeben sowie einen String, der den zu suchenden Bezeichner enthält. Zurück geliefert wird der Wert des Bezeichners als String oder `NULL`, wenn der Bezeichner nicht gefunden wurde.
Das nächste Modul heißt com_line.c. Es ist eine Sammlung von sieben Funktionen für die Kommandozeile, die drei triviale Funktionen enthält: die Funktion `print_Welcome()`, die einen einfachen Starthinweis ausgibt, die `print_Comline_Help()`, die über die Benutzung der Fassade einen kurzen Hilfebildschirm ausgibt sowie die Funktion `print_Comline_Error()`, die eine formatierte Fehlermeldung auf der Standardfehlerausgabe ausgibt. Die eigentliche Verarbeitung und Interpretation der Kommandozeilenargumente verrichtet die Funktion `checkingComline()`. In dieser Funktion wird als Erstes die Argumentenliste von `*argv[]` durchlaufen und auf Optionen durchsucht. Dazu wird in den Strings gesucht, ob das erste Zeichen ein Minus ist. Die folgenden Zeichen werden in einer `for`-Schleife so lange durchlaufen, bis der String endet. Die einzelnen Zeichen werden in einer `switch`-Anweisung ausgewertet. Trifft das Programm auf ein unbekanntes Zeichen, wird das Programm mit einer Fehlermeldung beendet. Trifft das Programm stattdessen auf die Zeichen „H“ oder „W“, wird eine Funktion aufgerufen, die versucht, eine Zahl aus dem String zu extrahieren. „H“ und „W“ sind die einzigen Kommandozeilenargumente, denen ein Wert folgen kann. Die Funktion zur Extraktion der Zahl heißt `extractValueFromString()`. Diese Funktion bekommt den String übergeben, die Position, von der ab die Zahl anfangen soll sowie die Information, wohin die Zahl gespeichert werden soll. Zurück geliefert wird die Anzahl der verarbeiteten Zeichen. Folgt beim Durchsuchen der Strings von `*argv[]` ein String, der kein Minuszeichen am Anfang enthält, dann ist es ein Dateiname. Das Durchsuchen und Interpretieren der Option endet an dieser Stelle. Nun wird ausgewertet, wie viele Dateinamen übergeben wurden. Wurden zu viele übergeben, endet das Programm mit einer Fehlermeldung. Zusätzlich wird mit der Funktion `checkFileExtension()`geprüft, ob der angegebene Dateiname die Endung „.LS“ enthält. Ist dies nicht der Fall, wird die Endung hinzugefügt. Am Ende der Funktion `checkingComline()` wird eine Funktion namens `checkingState()` aufgerufen. Diese Funktion prüft, ob die Angaben der Kommandozeile ausreichend und gültig sind.
Die beiden Dateien lparser.c und lparser.h bilden das Modul zur Interaktion mit dem Lparser-Softwarepaket. Unter anderem ist in ihnen auch eine Konvertierungsroutine enthalten, die LS-Dateien für die Version 4 in die Version 5 konvertiert. Diese Routine heißt `convertLP4toLP5()`. Sie benötigt den Namen der zu konvertierenden LS-Datei sowie den Dateinamen, in dem der konvertierte Inhalt gespeichert wird. Die Konvertierungsroutine ersetzt lediglich Symbole und führt keine Prüfung des Inhaltes durch. Sollten die Ziel- und Quelldatei den gleichen Namen haben, dann wird der konvertierte Inhalt in die Datei „temp.ls“ gespeichert und im Anschluss daran in den Namen der Zieldatei umbenannt. Die Funktion `Lparser()` ruft das Programm Lparser auf und leitet die Ausgabe in die Datei LPARSER.LOG um. Bevor die Fassade Lparser aufruft, wird die Funktion `Lparser()`aufgerufen. Diese wiederum ruft `checkingLSFile()` auf, die einen Plausibilitätstest der Werte vornimmt, weil Lparser selbst dazu nicht in der Lage ist. Hierfür wird die LS-Datei geöffnet und nach der ersten Zeile gesucht, die nicht nur aus einem Kommentar oder einer Leerzeile besteht. Dabei wird geprüft, ob die ersten drei Werte gültige Zahlen darstellen und ob der vierte Werte ein Axiom und keine Regel ist. Zusätzlich wird mit der Funktion `checkingRulesForBrackets()` geprüft, ob der linke oder rechte Kontext sowie der Bedingungsteil runde Klammern aufweisen. Nachdem die LS-Datei geprüft und Lparser ausgeführt wurden, wird die LPARSER.LOG Datei ausgewertet, um festzustellen, ob eine Regel als ungültig erklärt wurde. Danach endet die Aufgabe der Funktion `Lparser()`.

Eine weitere Funktion dieses Moduls ist `Lv2povid()`. Aufgabe dieser Funktion ist es, mit dem Programm LV2POVID zu kommunizieren. Bevor dies geschieht wird jedoch getestet, ob alle wichtigen Dateien vorhanden sind, in diesem Fall sind das die Dateien OUTPUT.INC, INFO.TXT und LV2POVID.CFG. Beim Aufruf von LV2POVID wird die Ausgabe in die Datei LV2POVID.LOG umgeleitet. Diese Datei wird nicht weiter ausgewertet. Eine weitere Routine in dem Modul ist `CorrectingPOVFile()`. Diese Funktion erstellt eine neue POV-Ray Datei mit den korrekten Daten für die Kameraposition, für die Kameraausrichtung und für die Lichtquellenposition. Der Inhalt der neuen POV-Datei ist identisch mit dem Inhalt von LV2POVID, korrigiert wurden lediglich die Daten . Um die Vektordaten aus der INFO.TXT auszulesen, gibt es noch die Funktion `extractVectorFromLine()`. Diese Funktion sucht nach einer öffnenden Klammer in einer gegebenen Zeile und versucht vom Anfang der geöffneten Klammer bis zur nächsten schließenden Klammer drei Werte zu extrahieren.
Das nächste Modul enthält zwei einzelne, nicht zusammengehörende Funktionen. Die Funktion `deleteFiles()` löscht alle Dateien, die zwischenzeitig erzeugt wurden, wie zum Beispiel die LOG-Dateien von Lparser oder LV2POVID. Darüber hinaus wird in der gesamten Fassade zum Löschen und Umbenennen von Dateien die Funktion `system()` verwendet. Entsprechend sind die Befehle, die `system()` an das Betriebssystem übergibt, betriebssystemabhängig. Zum Nachschlagen der Befehle für den Kommandozeileninterpreter von Windows sei auf [Mi93] verwiesen. Die Funktion `checkingEnvVariables()` prüft, ob eine Umgebungsvariable für QuietPOV angelegt wurde. Ist dies nicht der Fall, dann wird die Umgebungsvariable mit `SetEnvironmentVariable()` gesetzt. Dies ist eine der wenigen API-Funktionen von Windows. Sie wird verwendet, weil Windows 98 keine Umgebungsvariable mit der `system()`-Funktion setzten kann. Das Überprüfen der Umgebungsvariablen läuft wie folgt ab. Aus der Datei LPROCESS.CFG wird ein Pfad ausgelesen, mit dem angegeben wird, wo sich QuietPOV befindet. Nun werden nacheinander die Strings durchsucht, die in `*envp[]` enthalten sind. Dazu wird geprüft, ob die ersten vier Zeichen das Wort „Path“ ergeben. Wurde ein solcher String gefunden, wird dieser String Zeichen für Zeichen mit dem String aus der CFG-Datei verglichen. Taucht ein Zeichen, auf das nicht übereinstimmt, dann werden die Zeichen bis zum nächsten Semikolon übersprungen. Das Semikolon trennt die verschiedenen Werte der Umgebungsvariable „Path“.
Das letzte Modul beinhaltet die Funktionen für POV-Ray und QuietPOV. So startet `startPOV()` eine Instanz von POV-Ray, sofern noch keine Instanz von POV-Ray läuft. Dazu wird mit `FindWindow()` und dem Namen der POV-Ray Applikation nach einer bestehenden Instanz gesucht. Wurde der Wert Null zurückgeliefert, dann wird POV-Ray mit `WinExec()`gestartet, dabei wird angegeben, dass das Fenster versteckt werden soll. Wichtig ist in diesem Zusammenhang, dass `WinExec()` und `FindWindow()`API-Funktionen von Windows sind. Beendet wird POV-Ray mit `endPOV()`. Dazu wird erneut mit `FindWindow()` nach einer laufenden Instanz gefragt. Wurde eine gefunden, wird die Windows-Nachricht `WM_CLOSE` mit `SendMessage()`gesendet. Diese weist POV-Ray an, sich zu beenden. Auch `SendMessage()` ist eine API-Funktion von Windows. Zum Aufrufen von QuietPOV wird die Funktion `startQPOV()` verwendet. Diese ruft QuietPOV auf und leitet die Ausgabe in die Datei POV.LOG um. Bevor aber QuietPOV aufgerufen wird, muss die Fassade für POV-Ray noch eine INI-Datei erzeugen. Diese Aufgabe erfüllt die Funktion `createINI()`. Dabei werden alle POV-Ray relevanten Informationen aus der `InfoStruct` in die POV.INI Datei geschrieben, beispielsweise das Ausgabeformat oder die Größe der Grafik. Außerdem befindet sich in dem Modul noch eine Hilfsfunktion namens `getFilenameWithoutExt()`. Diese gibt den übergebenen Dateinamen, nur ohne die Endung, zurück. Sie wird verwendet, um für POV-Ray den Namen für die Ausgabedatei anzugeben.

5 Die graphische Oberfläche

5.1 Einführung zur graphischen Oberfläche

Auf den nächsten Seiten wird der Entwurf der graphischen Oberfläche dargestellt. Anders als bei der Fassade wird hier der Quelltext hier nur angerissen, weil der Schwerpunkt der Diplomarbeit liegt nicht bei der graphischen Oberfläche liegt, sondern auf der Fassade. Ziel der Applikation ist es, eine graphische Oberfläche für die Fassade herzustellen. Daher wird im Rahmen der Entwicklung der graphischen Oberfläche der Schwerpunkt auf Verständlichkeit und Einfachheit gelegt und nicht auf neue Funktionalität. Anzumerken ist jedoch, dass eine notwendige Voraussetzung für die Nutzung der graphischen Oberfläche ein Vorwissen zu L-Systemen und zur Turtle-Interpretation ist.

5.2 Die Entwicklungsumgebung für die graphische Oberfläche

Visual C++ 6.0 bietet mit der MFC eine objektorientierte Kapslung der Windows-API sowie eine Entwicklungsumgebung mit einer guten Möglichkeit zur Entwicklung von graphischen Oberflächen. Die Entwicklungszeit ist aber wesentlich länger als beim Rückgriff auf eine RAD-Entwicklungsumgebung (Rapid Application Development), die speziell für die Bedürfnisse der Entwicklung von graphischen Oberflächen angepasst wurde. Daher wird Delphi 6.0 als RAD-Entwicklungsumgebung ausgewählt.
Delphi ist eine Entwicklungsumgebung, die graphische Komponenten wie Buttons in der VCL (Visual Component Library) gekapselt hat und damit die Möglichkeit bietet, über Mausklicks Buttons zu erstellen und zu platzieren, ohne dabei einen Quellcode zu schreiben. Als Programmiersprache wird eine sehr stark weiterentwickelte Variante von Pascal verwendet, die sich Object Pascal nennt, und die eine sehr starke und leistungsfähige objektorientierte Sprache bildet. Ein weiterer Grund für den Einsatz von Delphi ist, dass es im objektorientierten Bereich sehr schnelle Programme erzeugen kann. Diese Programme sind in der Regel sogar schneller als C++ Anwendungen [SH03]. Object Pascal stellt zwar eine spezielle Weiterentwicklung von Borland dar, trotzdem können in Delphi entwickelte Programme auch unter anderen Betriebssystemen laufen. Denn Borland bietet ein Produkt namens Kylix an, das Delphi-Anwendungen problemlos kompiliert, wenn keine API-Funktionen von Windows verwendet werden. Damit die Oberfläche auf Linux schnell portiert werden kann, wurde bei der Entwicklung der graphischen Oberfläche darauf geachtet, möglichst wenige API-Funktionen von Windows zu verwenden.

5.3 Externe Bibliotheken und Komponenten

Obwohl Delphi eine RAD-Entwicklungsumgebung ist und ein sehr großes Sortiment an Komponenten und Bibliotheken enthält, gibt es dennoch Funktionen und Komponenten, die Delphi nicht enthält. Bei der Entwicklung der graphischen Oberfläche wird deshalb auf einige externe Komponenten und Bibliotheken zurückgegriffen. Dabei wurde darauf geachtet, dass diese mindestens Freeware, wenn nicht sogar Open Source sind. Diese externen Bestandteile werden hier kurz dargestellt. Dabei wird auch ihr Anteil bei der Entwicklung der Oberfläche verdeutlicht.

Die Standard-Grafikbibliothek von Delphi versteht zwar das Grafikformat Windows-Bitmap, aber nicht Targa. Da die Fassade auch Targa-Dateien erstellt, wird eine externe Bibliothek zur Erweiterung der Standard-Grafikbibliothek von Delphi benötigt. Mit [[Gra]] ist es möglich, das Standardobjekt `TPicture` so zu erweitern, dass es auch Targa-Grafiken lesen kann. Dadurch ist es möglich, mit jeder Komponente, die `TPicture` zum Laden von Grafiken benutzt, eine Targa-Datei anzuzeigen. Für die hier entwickelte Oberfläche wird die Komponente `TImage` verwendet.
Um das Verständnis für die Variablen zur Positionierung der Lichtquelle und der Kamera zu erleichtern, wird zur Visualisierung auf DirectX zurückgegriffen. Nun ist die DirectX-API als COM-Objekt ausgelegt und die eigentliche Verwendung für C/C++ bestimmt. Die direkte COM-Programmierung ist unter Delphi zwar möglich [Ko99], aber nicht unproblematische. Daher wäre eine Kapslung der DirectX-API für Delphi sehr wünschenswert. Realisiert wurde dies durch das JEDI Project (http://delphi-jedi.org/), dessen Ziele die Entwicklung von Bibliotheken und Komponenten ist sowie die Kapslung von API-Funktionen und deren vollständige Dokumentation. Damit verbunden ist die Absicht, die Möglichkeiten der Entwicklung mit Delphi zu erhöhen. Für die hier entwickelte graphische Oberfläche wird die Kapslung von DirectX 8 verwendet, weil für diese Version die meisten Informationen für die Entwicklung unter Delphi vorliegen. [[DX8]]
Das JEDI Project kapselt jedoch nicht nur die API von DirectX sondern auch die von Windows. Im Sammelsurium der visuellen Komponenten [[JVCL]] befindet sich auch eine Kapslung für die Windows-API `CreateProcess()`, die Komponente `JvCreateProcess`, mit dieser wird Lprocess für die graphische Oberfläche gestartet. Diese Komponente wurde verwendet, um den Rückgabewert von Lprocess auszuwerten und die Konsolenausgabe umzuleiten.

5.4 Entwurf der graphischen Oberfläche

In diesem Kapitel soll auf die Gestaltung der graphischen Oberfläche eingegangen werden und auf die damit verbundene Funktionalität. Dazu werden die relevanten Formulare (die Bezeichnung von Fenstern beim Entwurf) besprochen. Nicht besprochen werden Formulare, die keine direkte Funktionalität zur Oberfläche beisteuern, wie zum Beispiel das Info-Formular, die Formulare zur reinen Anzeige der Turtle-Kommandos, die Ausdrücke für den Bedingungsteil oder das Formular zur Anzeige der Konsolenausgabe. All diese Formulare zeigen lediglich Texte an und bieten nur die Möglichkeit der reinen Betrachtung an. Ein weiteres triviales Formular ist das Formular zur Anzeige des erzeugten Bildes. Auch dieses Formular ist zur reinen Betrachtung da und wird daher nicht im Einzelnen erläutert.

5.4.1 Das Hauptformular - TMDIMain

„Visual L“ ist eine MDI-Anwendung. MDI steht für „Multiple Document Interface“. Damit ist gemeint, dass die Anwendung ein Hauptfenster hat, in dem Kindfenster enthalten sein können. Dabei stellen die Kindfenster die eigentliche Anwendung dar. Das Hauptfenster bildet den gemeinsamen Rahmen für alle Kindfenster. Durch das MDI-Konzept ist es möglich, mehrere Instanzen einer Anwendung aufzurufen. Das Gegenstück zu MDI ist SDI, dies steht für „Single Document Interface“. In einem SDI-Programm existiert nur ein Fenster für die Anwendung. Der Mehraufwand einer MDI-Anwendung ist gerade bei Anwendungen, die Editor-Funktionen realisieren sollen, gerechtfertig.

Denn sie ermöglicht es dem Benutzer, mehrere Dokumente zu öffnen, diese zu vergleichen bzw. abzugleichen oder ein Dokument durch ein anderes zu ergänzen, ohne dabei zwei Instanzen einer Anwendung zu starten. In Abbildung 5.1 ist das Hauptformular der graphischen Oberfläche zu sehen.
Visuell besteht das Hauptformular nur aus einer Menüleiste, einer Symbolleiste und einer Statusleiste. Um das Einarbeiten in die Applikation zu erleichtern, wird Wert darauf gelegt, übliche Menüpunkte, die man bei einer MDI-Anwendung erwartet, zu integrieren und deren Reihenfolge und Bezeichnung beizubehalten. In die Menüleiste werden die Menüpunkte aufgenommen, die am meisten vom Anwender benutzt werden. Die Statusleiste gibt Informationen zu dem Element aus, über dem sich gerade der Mauszeiger befindet. Hinter der Gestaltung steht folgenden Zielsetzung: das Hauptformular darf nicht zu überladen wirken, nicht zu Verwirrungen führen und dem Anwender einen schnellen Einstieg bieten.

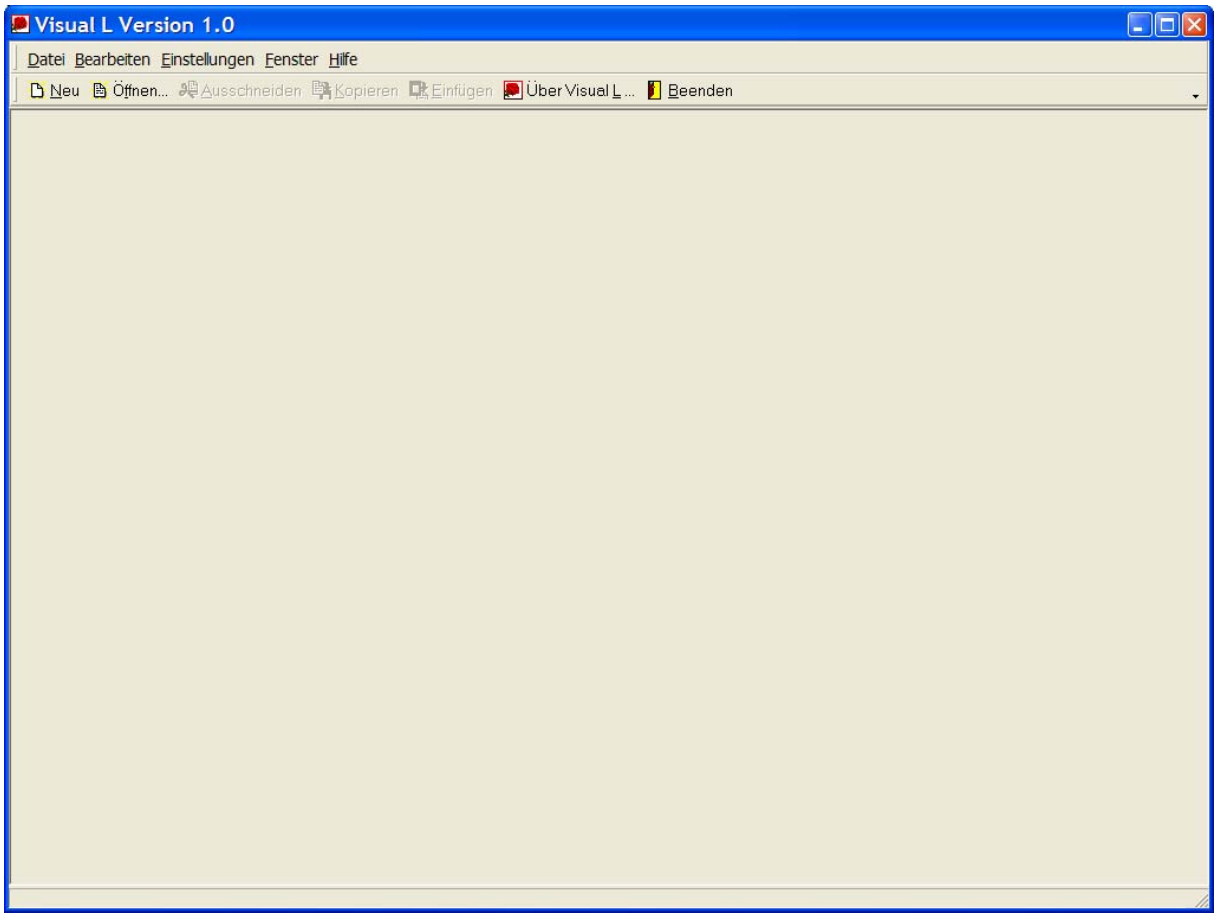


Abbildung 5.1: Das Hauptfenster von „Visual L“

Wenn eine LS-Datei importiert wird, dann wird der Inhalt der Datei an Lprocess weitergegeben, um den Inhalt zu konvertieren. Daher wird auch vorher gefragt, ob der Inhalt in die gleiche Datei oder in einer anderen Datei gespeichert werden soll. Sollte Lprocess mit einem Fehler bei der Konvertierung terminieren, wird die Konsolenausgabe in einem separaten Fenster ausgegeben, damit der Benutzer sich die Fehlermeldung anschauen kann. Nach dem Konvertieren wird der Inhalt der LS-Datei geprüft und alle gültigen Teile werden in die Komponenten eines Kindfensters übertragen. Sollte ein Teil nicht gültig sein, wird in der unteren `TListBox` Komponente des Kindfensters eine Nachricht eingefügt, welcher Teil nicht übernommen wurde und warum. So erscheint zum Beispiel eine Nachricht bei einem ungültigen oder fehlenden Wert für die Rekursionstiefe, den Basiswinkel, die Basisstärke oder für das Axiom.

Fehlerhafte Produktionen sind solche, die zum Beispiel das Symbol für einen linken oder rechten Kontext haben, aber keinen Inhalt aufweisen. Beim Öffnen einer bestehenden LS-Datei in Version 5 wird die gleiche Prüfung durchgeführt. Beim Start der Anwendung wird außerdem der Inhalt der CFG-Datei ausgelesen und in einer Struktur gespeichert, die für alle weiteren Teile der Anwendung zugänglich ist. Sollte die CFG-Datei nicht bestehen, wird diese durch eine CFG-Datei mit Standardwerten ersetzt. Sollte ein Wert in einer CFG-Datei ungültig sein, so wird ebenfalls ein Standardwert verwendet. Beim Beenden der Anwendung wird der Inhalt der Struktur wieder zurück in die CFG-Datei geschrieben. Außerdem wird mit Hilfe von Lprocess eine eventuell gestartete Instanz von POV-Ray beendet.

5.4.2 Das Kindfenster - TMDIChildMain

Die eigentliche Anwendung spielt sich bei einer MDI-Anwendung im Kindfenster ab. Das Kindfenster kann in zwei Teile aufgeteilt werden. In Abbildung 5.2 wird das Kindfenster dargestellt.

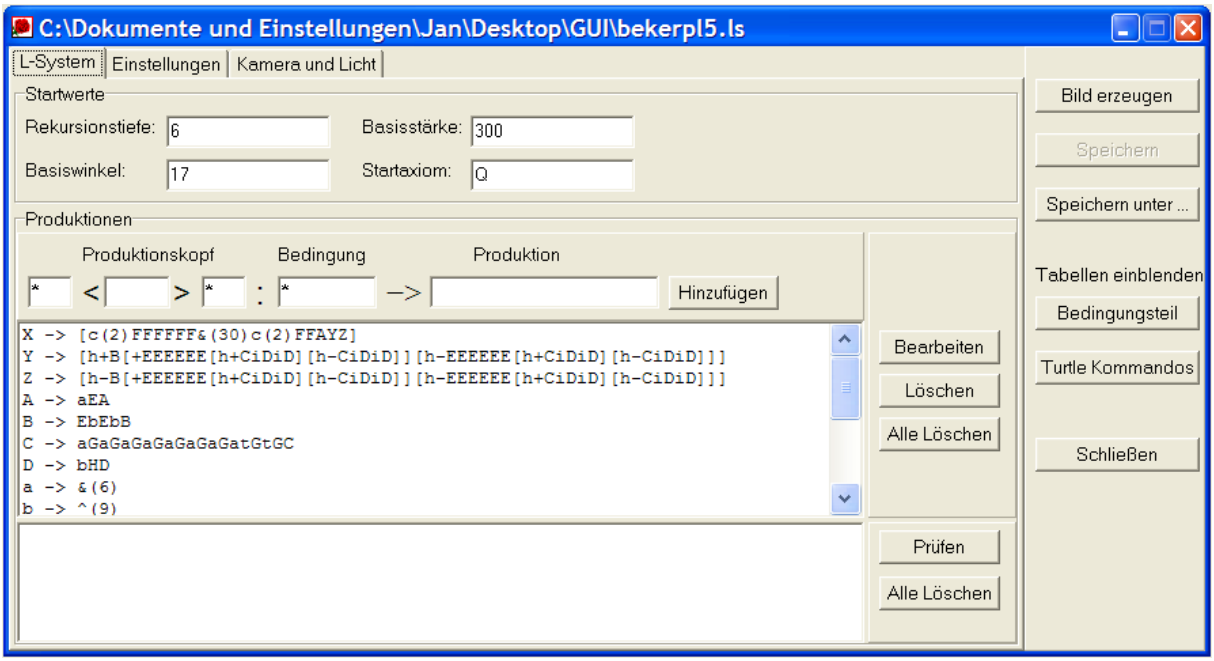


Abbildung 5.2: Das Kindfenster von „Visual L“

Links befindet sich der Teil, in dem das L-System editiert wird und in dem seine relevanten Einstellungen vorgenommen werden. Um nicht die Übersicht zu verlieren, werden drei Reiter verwendet. Auf dem ersten wird das L-System definiert und editiert. So existieren Editierfelder, in denen die Rekursionstiefe, der Basiswinkel, die Basisstärke und das Startaxiom eingegeben werden können. Außerdem wurde eine Eingabemaske für die Definition einer Produktion erstellt. Viele Applikationen zu L-Systemen besitzen einen einfachen Text-Editor, in dem eine strenge Abfolge eingehalten werden muss, um ein gültiges L-System zu definieren [[L3D]] [[LS4]]. Für den Benutzer ist nicht auf Anhieb ersichtlich, wie das umgesetzt werden kann. Um dieser Schwäche entgegenzuwirken und um die Syntax und das Verständnis für die Definition eines L-Systems zu erhöhen, werden Editierfelder und die Eingabemaske erstellt. Wird eine Produktion hinzugefügt, wie beim Öffnen einer LS-Datei, so wird die Produktion daraufhin überprüft, ob die Teile der Produktion auch richtig angegeben wurden.

Ist das nicht der Fall, gibt es beispielsweise bei einer Produktion einen Bedingungsteil aber keine Parameter im Produktionskopf, dann wird diese nicht angenommen. Zusätzlich dazu kann das L-System geprüft werden, indem auf den Button „Prüfen“ geklickt wird. Dabei wird kontrolliert, ob die Anzahl der schließenden und öffnenden, eckigen und geschweiften Klammern in einer Produktion gleich sind. Ebenfalls kontrolliert wird, ob für ein Symbol mehrere oder keine Produktion definiert wurde und ob die im Produktionskopf definierten Parameter verwendet wurden oder solche, die nicht definiert wurden. Auf dem zweiten Reiter können einige Grafikparameter eingestellt werden, beispielsweise ob Anti-Aliasing verwendet werden soll, welche Höhe und Breite die Grafik haben soll und ob das Grafikformat Targa oder Windows-Bitmap sein soll. Auf dem letzten Reiter kann die relative Position der Kamera zum Objekt bestimmt werden sowie die relative Position der Lichtquelle zur Kamera. Dazu kann über sechs `TTrackBar` Komponenten die Einstellung des x, y und z-Wertes vorgenommen werden. Um sich die Auswirkungen zu verdeutlichen kann ein Fenster eingeblendet werden, auf dem ein 3D-Kubus zu sehen ist. Entsprechend der Einstellung der `TTrackBar` Komponenten werden die Kamera und das Licht umpositioniert.
Die rechte Seite des Kindfensters besteht lediglich aus sechs `TButton` Komponenten. Unter anderem kann mit den Buttons „Bedingungsteil“ und „Turtle Kommandos“ ein Fenster eingeblendet werden, in dem die Turtle-Kommandos und die erlaubten Zeichen für gültige Ausdrücke im Bedingungsteil angezeigt werden. Der Button mit der Aufschrift „Schließen“ sollte selbsterklärend sein, ebenso wie die Buttons „Speichern“ und „Speichern unter ...“. Wird auf den Button „Bild erzeugen“ geklickt, dann wird als Erstes das L-System geprüft. Diese Prüfung entspricht der Prüfung die erfolgt, wenn auf den Button „Prüfen“ geklickt wird. All diese Prüfungen sind nur Warnungen, das heißt sie liefern nur Hinweise auf unerwartete Ergebnisse, es kann trotzdem ein Bild erzeugt werden. Nach der Prüfung werden die aktuellen Einstellungen in der CFG-Datei gespeichert, damit Lprocess diese Parameter anwenden kann. Anschließend wird über die `JvCreateProcess` Komponente Lprocess gestartet. Tritt während der Verarbeitung ein Fehler auf, wird die Konsolenausgabe dem Benutzer in einem separaten Fenster zugänglich gemacht. Im Normalfall wird am Ende der Bilderzeugung ein Fenster geöffnet, in dem das Bild angezeigt wird. Während der Bilderzeugung ist die Anwendung aktiv, das heißt der Anwender kann weiterarbeiten, während er auf das Bild wartet.

5.4.3 Einstellungs-Dialog - TCFGForm

Um den Inhalt der CFG-Datei ohne das Öffnen eines Kindfensters einzeln bearbeiten zu können, wird ein Dialog-Formular namens `TCFGForm` angelegt. Dieses Dialog-Formular ist in Abbildung 5.3 zu sehen.
Seine Einstellungsmöglichkeiten sind fast identisch mit denen des Kindfensters. Das Formular enthält zwei Reiter, einen für allgemeine Einstellungen und einen anderen für die Kamera und das Licht. Im Unterschied zum Kindfenster wurden aber zwei Editierfelder hinzugefügt, um den Pfad für POV-Ray und QuietPOV anzugeben. Alternativ kann das Programm auch nach dem Pfad von POV-Ray und QuietPOV suchen.
Das Dialog-Formular wird modal angezeigt, das heißt solange das Dialog offen ist, kann nicht mit der Applikation weiter gearbeitet werden.

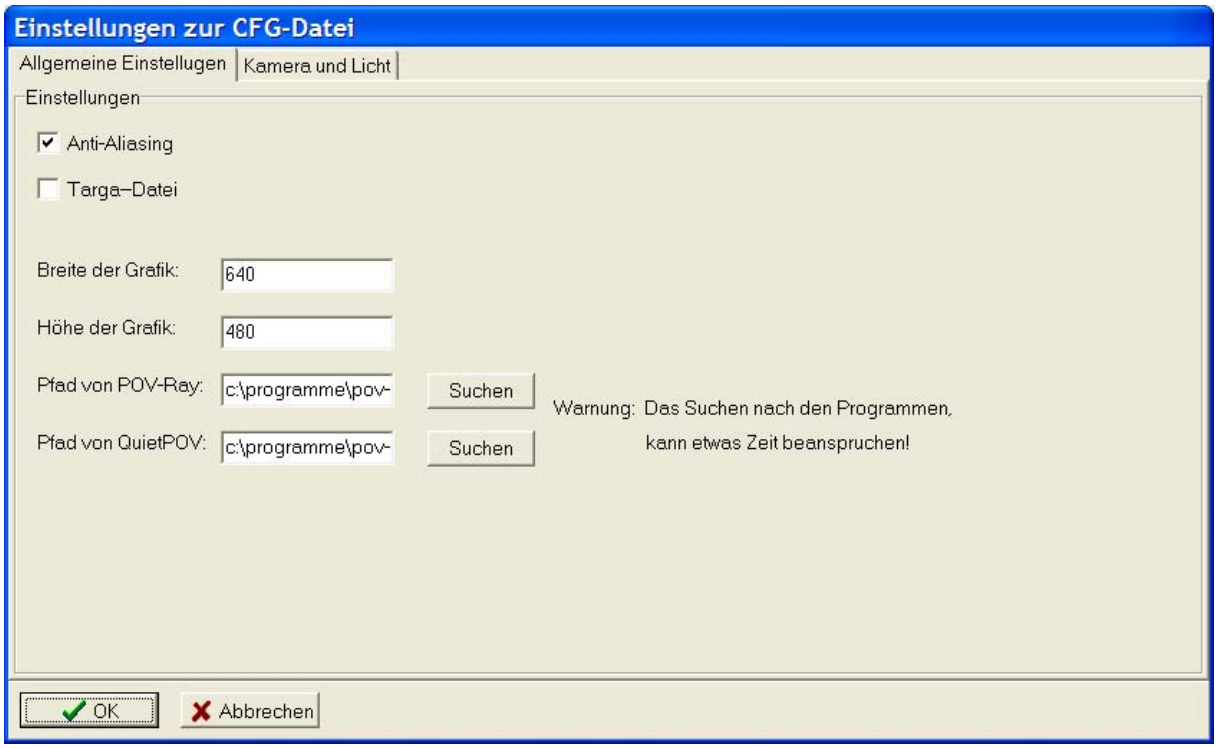


Abbildung 5.3: Das Dialog-Fenster zum Editieren der CFG-Datei

5.4.4 Die 3D-Ansicht - TDirect3DForm

Die 3D-Ansicht zur Visualisierung der relativen Position von Kamera und Lichtquelle wurde mit DirectX 8 realisiert. In Abbildung 5.4 ist eine beispielhafte Darstellung dieser beiden Einstellungen zu sehen.
Die ausführliche Beschreibung der Entwicklung einer Anwendung mit DirectX kann an dieser Stelle nicht gegeben werden, weil hier nur sehr allgemein auf die Programmierung eingegangen wird. Daher sei zur DirectX-Programmierung auf folgende Quelle verwiesen [Rou01] [SDU99] [Du00] [EG01]. Speziell zur Entwicklung von DirectX-Anwendungen mit Delphi sei auf [Ra04] verwiesen.
Mit der Einführung von Windows 95 wollte Microsoft die DOS-Ebene verlassen und die Entwickler motivieren, Windows-Anwendungen zu entwickeln. Dazu zählt auch, dass künftige Spiele nicht mehr unter DOS sondern unter Windows laufen sollten. Das Problem war damals, dass Windows keine Unterstützung für aufwendige Grafikprogrammierung bot. Daher hat Microsoft 1995 die WinG (Windows Graphics) Bibliothek eingeführt. Diese war der Vorläufer von DirectX (vgl. dazu [Ras95]). Aber WinG konnte sich nicht durchsetzen. Deshalb wurde die Bibliothek vollkommen überarbeitet, stark erweitert und als DirectX veröffentlich. DirectX teilt sich dabei in mehrere Teile auf, in DirectPlay, DirectDraw und DirectSound. Ein weiterer Teil heißt Direct3D. Er ist für das Programmieren von 3D-Anwendungen gedacht. Wie alle Komponenten von DirectX basiert auch Direct3D auf der COM-Technologie.
Im Gegensatz zur Modellierung mit einem Renderer kennt Direct3D als 3D-Primitive nur Dreiecke, die aus drei Vertex-Punkten bestehen. Hintergrund dafür ist die Hardware der 3D-Grafikkarten und der darauf befindlichen Vertex-Shader. Um einen Kubus unter Direct3D zu visualisieren, muss der Kubus in Dreiecke zerlegt werden. Da für jede Seite zwei Dreiecke benötigt werden, kommt man auf zwölf Dreiecke und auf 36 Vertex-Punkte, weil jedes Dreieck aus drei Vertex-Punkten besteht. Nun muss eine Datenstruktur angelegt werden, in der die 36 Vertex-Punkte gespeichert werden können.

Microsoft bietet dazu das FVF (Flexible Vertex Format) an. Über die Konstante `D3D8T_CUSTOMVERTEX` kann eine flexible Datenstruktur für Vertex-Punkte festgelegt werden. So benötigt man neben der x-, y- und z-Koordinate noch die Farbe und den Normalen-Vektor für jeden Vertex-Punkt.
Bevor die 3D-Szene von Direct3D gerendert werden kann, müssen einige Funktionen die Szene initialisieren und vorbereiten. So werden in der Prozedur `D3DInit` die gesamten Komponenten zur Kommunikation und Erzeugung der 3D-Szene initialisiert. In der Prozedur `D3DInitScene` wird die 3D-Szene vorbereitet. An dieser Stelle werden Lichtquelle und Kamera positioniert und bei der Lichtquelle werden diverse Einstellungen festgelegt, beispielsweise der Typ der Lichtquelle und deren Reichweite. Um die Szene richtig beleuchten zu können, muss den Dreiecken noch ein Material zugewiesen werden. Über die Datenstruktur des Materials kann festgelegt werden, ob das Objekt das Licht zum Beispiel stärker oder schwächer reflektieren soll. Im Anschluss kann in der Prozedur `D3DRender` die 3D-Szene beschrieben werden. Dazu wird zwischen den Anweisungen `BeginScene` und `EndScene` der Quelltext zur Beschreibung der 3D-Szene hinterlegt. Jegliche Transformation der 3D-Objekte wird an dieser Stelle verrichtet, auch das komplette Laden der Vertex-Punkte. Damit Direct3D beim Beenden der Anwendung ordentlich beendet werden kann, müssen zwei weitere Prozeduren definiert und ausgeführt werden. In der Prozedur `D3DKillScene` muss der Speicher aller Vertex-Listen wieder freigegeben werden und in der Prozedur `D3DShutdown` müssen alle Variablen gelöst werden, die mit den COM-Objekten von Direct3D verbunden sind.

Abbildung 5.4: Die 3D-Ansicht mit Direct3D

6 Tests und Auswertung der entwickelten Software

Im folgenden Kapitel werden die beiden Applikationen Fassade und graphische Oberfläche auf ihre Tauglichkeit getestet. Dazu werden beide Anwendungen getrennt betrachtet. Die Tests beziehen sich größtenteils auf Software-Tests, das heißt es wird geprüft, ob die Programme zuverlässig laufen und wie sie auf widrige Umstände reagieren.

6.1 Test der Fassade

6.1.1 Testumgebung

Die Tests für die Fassade wird auf einem Intel „Pentium 4 Mobil“ mit 2,5 GHz, 512 MB RAM und einer NVIDIA „GeForce4 4200 Go“ Grafikkarte durchgeführt. Als Betriebssystem wird Microsoft Windows XP mit Service Pack 1 verwendet. Da der Computer ein Notebook ist, wird das Notebook während der Testläufe stationär an eine Steckdose angeschlossen. Somit laufen die Komponenten des Notebooks nicht im Energiesparmodus, sondern im Desktopmodus und nutzen ihre volle Performance.

6.1.2 Testumfang

Die Fassade wird auf drei Aspekte geprüft, auf die Performance, die Belastung und auf die Funktionalität.
Beim Performancetest wird neben der Abarbeitungszeit noch die Frage geklärt, ob die Fassade performanter ist, wenn die Anwendung als Release-Version kompiliert wurde. Außerdem wird aufgezeigt, wie zeitintensiv welche Teile der Fassade sind.
Der Belastungstest läuft parallel zum Performancetest. Mit ihm soll die Prozessor-, Speicher- und Festplattenbelastung aufgezeigt werden. Außerdem wird überprüft, ob die Anwendung unter Belastung zuverlässig läuft.
Zum Abschluss wird noch getestet, wie sich die Anwendung verhält bei unerwarteten Werten von der Kommandozeile, von der CFG-Datei oder von der LS-Datei.

6.1.3 Testdurchführung

Für den Performancetest wurde eine Batch-Datei angelegt. In dieser wird am Anfang und am Ende die aktuelle Systemzeit ausgegeben. Dazwischen wird die gerade zu testende Anwendung mit verschiedenen Dateien aufgerufen. Für jede Anwendung wird der Test mehrmals durchgeführt, um das arithmetische Mittel berechnen zu können. Der Performancetest wird mit der gleichen Batch-Datei und dem gleichen Inhalt auf die Release- und Debug-Version des Lparsers angewendet, um den Leistungsunterschied zu zeigen.
Zum Aufzeichnen der Belastung des Systems während des Performancetests wird der Windows XP eigenen Leistungs-Monitor verwendet. Dabei werden die Prozessorzeit, die durchschnittliche Warteschlangenlänge des physikalischen Datenträgers sowie die angeforderten Seiten des Arbeitsspeichers pro Sekunde beobachtet.

Es sei darauf hingewiesen, dass bei einem Software-Monitor ein Teil der Performance an das Monitor-Programm verloren geht und dass die Ergebnisse daher nicht der reinen Abarbeitungszeit entsprechen. Zusätzlich wird das Ergebnis des Performancetests und des Belastungstests von Prozessen und Programmen im Hintergrund beeinflusst. Diese weiteren Prozesse und Programme sind wiederum abhängig vom jeweiligen individuell installierten System.
Viele Fehler geschehen bei der Eingabe. Deshalb wird für den Funktionstest der Fassade die Schnittstellen getestet. Dabei werden auch die Grenzwerte der Argumentenverarbeitung von der Kommandozeile ausgetestet. Außerdem werden die Grenzwerte für das Auslesen und Verarbeiten der CFG- und LS-Datei auf ihre Tauglichkeit hin geprüft, indem sie mit ungültigem Inhalt gefüllt werden. Die Fassade selbst wird über die Eingabeaufforderung aufgerufen und nicht über ein externes Programm.

6.1.4 Auswertung

6.1.4.1 Performancetest

Die Auswertung erfolgt in der Reihenfolge der Abarbeitung der Fassade. Der erste Verarbeitungsschritt ist die Konvertierung der LS-Datei in die Version 5. Konvertiert werden fünfmal 117 Dateien. Das Ergebnis ist in Tabelle 6.1 zu sehen. Die Dateien sind in der Regel klein und das Ersetzen einzelner Zeichen benötigt nicht viel Zeit.

1. Durchlauf	2. Durchlauf	3. Durchlauf	4. Durchlauf	5. Durchlauf	Mittelwert
3 Sek.	3 Sek.	3 Sek.	3 Sek.	4 Sek.	3:20 Sek.

Tabelle 6.1: Zeit zum Konvertieren von 117 Dateien

Der nächste Schritt ist die Verarbeitung der Dateien durch Lparser. Dazu werden zehnmal 101 Dateien zur Verarbeitung an den Lparser weitergegeben. Dabei wurde der Lparser fünfmal in der kompilierten Debug-Version und fünfmal in der Release-Version verwendet. Interessant ist, dass die Debug-Version gegenüber der Release-Version zwar dreimal größer ist, aber auch fast dreimal schneller. Wegen der extremen Langsamkeit von Lparser werden 6 Dateien nicht verwendet, weil sie eine zu lange Verarbeitungszeit (mehrere Stunden oder Tage) benötigen würden.

1. Durchlauf	2. Durchlauf	3. Durchlauf	4. Durchlauf	5. Durchlauf	Mittelwert
3:48 Min.	3:49 Min.	3:50 Min.	3:50 Min.	3:54 Min.	3:50:20 Min.

Tabelle 6.2: Zeit für die Verarbeitung von 101 Dateien vom Lparser in der Debug-Version

1. Durchlauf	2. Durchlauf	3. Durchlauf	4. Durchlauf	5. Durchlauf	Mittelwert
10:01 Min.	9:58 Min.	9:59 Min.	10:00 Min.	9:59 Min.	9:59:40 Min.

Tabelle 6.3: Zeit für die Verarbeitung von 101 Dateien vom Lparser in der Release-Version

Nun werden die 3D-Szenen von Lparser mit dem Programm LV2POVID in POV-Ray Szenen transformiert. Dazu werden 95 Dateien fünfmal von LV2POVID verarbeitet. Aus nicht ersichtlichen Gründen hat LV2POVID sechs Dateien aus dem vorherigen Schritt nicht verarbeitet können.

1. Durchlauf	2. Durchlauf	3. Durchlauf	4. Durchlauf	5. Durchlauf	Mittelwert
2:23 Min.	2:23 Min.	2:22 Min.	2:23 Min.	2:23 Min.	2:22:80 Min.

Tabelle 6.4: Zeit für das Transformieren von 95 Dateien

Der letzte Schritt ist die Erzeugung der 3D-Grafiken. Dazu wird POV-Ray vorher gestartet, wobei die Befehle über QuietPOV gesendet werden. Der Test wird mit 95 Dateien zehnmal durchgeführt, fünfmal ohne Anti-Aliasing und fünfmal mit Anti-Aliasing.

1. Durchlauf	2. Durchlauf	3. Durchlauf	4. Durchlauf	5. Durchlauf	Mittelwert
8:02 Min.	7:47 Min.	7:59 Min.	8:00 Min.	8:00 Min.	7:57:60 Min.

Tabelle 6.5: Zeit für das Erzeugen von 95 Grafiken ohne Anti-Aliasing

1. Durchlauf	2. Durchlauf	3. Durchlauf	4. Durchlauf	5. Durchlauf	Mittelwert
12:33 Min.	12:25 Min.	12:36 Min.	12:14 Min.	12:23 Min.	12:26:20 Min.

Tabelle 6.6: Zeit für das Erzeugen von 95 Grafiken mit Anti-Aliasing

Nun wird der komplette Verarbeitungszyklus mit Lprocess durchlaufen. Dabei werden 95 Dateien fünfmal verarbeitet. Alle Dateien waren schon konvertiert und es wurde kein Anti-Aliasing verwendet. Interessanterweise lässt sich die Release-Version von Lprocess zwar kompilieren, aber unter Windows XP auch dann nicht ausführen, wenn es unter Windows XP kompiliert wurde. Daher wurde nur die Debug-Version getestet. Vom Lparser wird die Debug-Version für diesen Test verwendet.

1. Durchlauf	2. Durchlauf	3. Durchlauf	4. Durchlauf	5. Durchlauf	Mittelwert
14:46 Min.	14:20 Min.	14:26 Min.	14:32 Min.	14:11 Min.	14:27:00 Min.

Tabelle 6.7: Zeit für die Verarbeitung von 95 Dateien mit Lprocess

Wenn nun die Verarbeitungszeit vom Lparser auf 95 Dateien normalisiert wird, wird folgende Zeitaufteilung sichtbar.

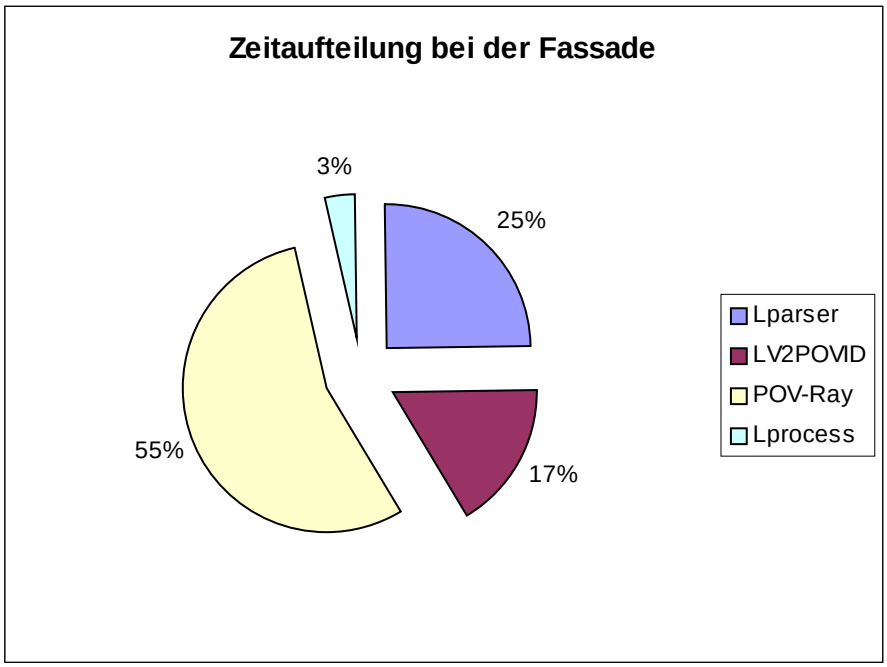


Abbildung 6.1: Tortendiagramm der Zeitaufteilung

6.1.4.2 Belastungstest

In Tabelle 6.8 ist die Belastung der Komponenten Prozessor, Speicher und Datenträger für die Anwendungen Lparser, LV2POVID, POV-Ray und Lprocess (der komplette Verarbeitungszyklus) angegeben. Dabei ist der Wertebereich normalisiert von 0 (Minimum) bis 100 (Maximum). Die Daten wurden parallel zum Performancetest aufgenommen und liegen daher auch fünfmal vor. Alle Werte sind dementsprechend arithmetische Mittelwerte.

Programm	Prozessor	Speicher	Datenträger
Lparser (Debug)	99,80	0,00	1,00
Lparser (Release)	100,00	0,00	0,00
LV2POVID	99,00	2,60	0,00
POV-Ray (ohne Anti-Aliasing)	77,00	0,00	0,00
POV-Ray (mit Anti-Aliasing)	84,80	0,00	0,00
Lprocess	87,00	0,20	0,00

Tabelle 6.8: Ergebnis der Belastungen der einzelnen Komponenten

Während der Batch-Abarbeitung ist kein Programm aus Belastungsgründen abgestürzt, hat nicht mehr reagiert oder ist mit zunehmenden Durchläufen langsamer geworden. Wie der Tabelle zu entnehmen ist, ist der Prozessor der größte Belastungsfaktor. Der Speicher wird lediglich am Anfang kurzfristig stärker beansprucht.

6.1.4.3 Funktionstest

Kommandozeile:
Das Programm wird nicht ausgeführt, wenn eine ungültige Kombination von Optionen vorliegt, wie zum Beispiel „-eE“.

Denn dies würde bedeuten, dass POV-Ray sofort und nach Beendigung der Verarbeitung beendet werden soll. Auch wenn Optionen doppelt erscheinen und eine ungültige Kombination darstellen, wird das Programm nicht ausgeführt. Wird jedoch die Option für die Bildhöhe zweimal angegeben, dann wird der zweite Wert verwendet. Ist beim zweiten Erscheinen der Option kein Wert angegeben, dann wird das Programm mit einer Fehlermeldung beendet. Das Programm terminiert ebenfalls ordentlich mit einer Fehlermeldung, wenn nach der Option für die Bildhöhe kein Wert folgt, nicht definierte Zeichen verwendet werden oder Zahlen an nicht erlaubter Stelle erscheinen.
Werden zwei Dateinamen angegeben, aber die Option zur Konvertierung nicht gesetzt, dann wird immer der zweite Dateiname zur Verarbeitung verwendet. Entsprechend terminiert das Programm mit einer Fehlermeldung, wenn die Datei nicht existiert. Im Übrigen dürfen die Dateinamen keine Leerzeichen enthalten, weil dies vom Kommandozeileninterpreter als zwei getrennte Argumente interpretiert wird.
Bei der Reihenfolge der Optionen und Dateinamen müssen die Optionen immer an erster Stelle angegeben werden. Sollten Optionen nach einem Dateinamen gefunden werden oder mehr als zwei Dateinamen, dann wird das Programm mit einer Fehlermeldung beendet. Sollte das Programm nur auf die Option „-e“ (POV-Ray sofort beenden) treffen, gefolgt von einem Dateinamen, dann wird POV-Ray beendet und der Dateiname ignoriert.
Die Länge der Zeichenkette für die Option kann beliebig lang sein, aber nicht die Zeichenkette für einen Dateinamen. In der Datei globals.h kann die maximale Länge für die Zeichenkette eines Dateinamens bestimmt werden, wird diese Länge überschritten, stürzt das Programm ab. Soll mit sehr langen Dateinamen gearbeitet werden, kann der Wert für die maximale Dateinamenlänge in der globals.h (Standardwert sind 50 Zeichen) geändert und die Fassade neu kompiliert werden.

CFG-Datei:
Liegt keine Datei mit dem Namen LPROCESS.CFG vor, wird eine Datei mit Standardwert erzeugt. Die Optionen innerhalb der CFG-Datei können in einer willkürlichen Reihenfolge vorliegen. Sollte eine Option nicht komplett vorhanden sein oder für eine Option kein Wert vorliegen, dann wird die Fassade mit einer Fehlermeldung beendet. Ebenfalls wird das Programm mit einer Fehlermeldung beendet, wenn ein nicht erwarteter Wert eingelesen wurde, wie zum Beispiel ein „aaa“, statt ein „on“ bei der Option „Files“. Die Fassade terminiert mit einer Fehlermeldung, wenn sich POV-Ray oder QuietPOV nicht in dem Verzeichnis befinden, das in der CFG-Datei hinterlegt wurde. Die Kommentare in der CFG-Datei müssen sich in einer separaten Zeile befinden, ansonsten wird dem Wert der Option der Kommentar zur Weiterverarbeitung hinzugefügt.
Sollte die maximale Länge für eine Zeile in der CFG-Datei überschritten werden, dann wird das Programm mit einer Fehlermeldung beendet. Da sich dieser Überlauf erst in der nächsten Zeile bemerkbar macht, wird in der Fehlermeldung die Folgezeile als Fehlerquelle angegeben. Der Standardwert für die Länge einer Zeile beträgt 150 Zeichen. Dies ist in der Datei globals.h definiert. Sollten längere Zeilen benötigt werden, muss der Wert verändert und die Fassade neu kompiliert werden.

LS-Datei:
Fehlen die Parameter für Rekursionstiefe, Basiswinkel, Basisstärke und Startaxiom, dann wird das Programm mit einer entsprechenden Fehlermeldung beendet. Sollte eine Datei eine Zeile enthalten, die mehr als 1000 Zeichen enthält, stürzt Lparser ab. Die Fassade selbst terminiert mit der Fehlermeldung, dass keine „output.inc“ für LV2POVID vorliegt und beendet sich ordnungsgemäß, obwohl Lparser abgestürzt ist. Um die Länge einer Zeile zu ändern, muss im Quelltext von Lparser eine Präprozessor-Anweisung geändert und Lparser neu kompiliert werden.

6.2 Test der graphischen Oberfläche

6.2.1 Testumfang

Der Test der graphischen Oberfläche teilt sich in zwei Bereiche auf. Erstens soll die Funktionalität der Software getestet werden und zweitens eine Usability-Studie über die Anwendbarkeit der graphischen Oberfläche präsentiert werden.
Der Funktionstest soll im laufenden Betrieb zeigen, wie die Anwendung auf Fehleingaben reagiert, bzw. wie sie auf Probleme mit den Inhalten von LS-Dateien und / oder mit dem Inhalt in der CFG-Datei umgeht.
Für die Usability-Studie werden freiwillige Personen gesucht. Diese werden mit dem theoretischen Wissen zu L-Systemen ausgestattet und im Anschluss daran mit der Oberfläche konfrontiert. Im Anschluss daran sollen sie einfach kleine Aufgaben lösen.

6.2.2 Testdurchführung

Beim Funktionstest werden die Inhalte von LS-Dateien und der Inhalt der CFG-Datei manipuliert, um Fehler zu provozieren und die Reaktion des Programms zu beobachten. Seitens der Oberfläche werden bei den Eingabefeldern und Schiebereglern die Grenzwerte ausgetestet.
Bei der Usability-Studie werden die Personen mit den theoretischen Grundlagen der L-Systeme vertraut gemacht. Dazu wird der Vorgang der Ableitung bei einem L-System erläutert. Während dessen werden notwendige Begriffe eingeführt, wie zum Beispiel die Begriffe Rekursionstiefe und Produktionen. Zuletzt wird die Turtle-Interpretation erläutert und mit wenigen einfachen Kommandos illustriert. Nach dem theoretischen Teil sollen die Personen kleine Aufgaben lösen. So soll zum Beispiel ein bestehendes L-System geladen und das Bild erzeugt werden oder ein neues Kindfenster geladen werden sowie die Felder für Rekursionstiefe, Basiswinkel, Basisstärke und Startaxiom exemplarisch ausgefüllt werden. Zusätzlich dazu soll noch eine Produktion eingegeben werden und die Personen sollen sich den 3D-Kubus anschauen und mit den Schiebereglern die Werte für die Kamera- und Lichtquellenposition verändern. Dabei sollen sie kommentieren, ob das Ergebnis zu erwarten war.

6.2.3 Auswertung

6.2.3.1 Funktionstest

Wird beim Öffnen einer LS-Datei ein Fehler entdeckt, wird dies im Kindfenster in der Nachrichtenliste eingefügt. Alle anderen nicht fehlerhaften Teile werden in die entsprechenden Komponenten des Kindfensters eingetragen. Ein Fehler ist beispielsweise, wenn die Rekursionstiefe, der Basiswinkel, die Basisstärke oder das Startaxiom fehlen. Ein weiterer Fehler ist, wenn eine Zeile länger als 1000 Zeichen ist. Seitens der Produktionen wird ein Fehler ausgegeben, wenn der Produktionskopf oder -körper fehlt oder wenn zwar eine Klammer für den linken oder rechten Kontext vorhanden aber ohne Inhalt ist. Weiterhin als Fehler deklariert ist, wenn im Produktionskopf ein Klammer-Ausdruck steht, dieser aber fehlerhaft ist, zum Beispiel wenn Sonderzeichen in der Parameterliste vorhanden sind oder wenn sich hinter der rechten Klammer noch Zeichen befinden. Falls ein Bedingungsteil vorhanden ist, wird dieser daraufhin geprüft, ob ungültige Zeichen vorliegen, beispielsweise Klammern.

Die gleiche Prüfung wird im Kindfenster jedes Mal durchlaufen, wenn eine neue Produktion hinzugefügt wird.
Das Programm reagiert auf das Nichtvorhandensein der CFG-Datei, indem eine neue mit Standardwerten erzeugt wird. Wurde in einer CFG-Datei ein Wert nicht gefunden, wird der Option ein Standardwert zugewiesen.
Wird im Formular `TCFGForm` auf den Button „Suchen“ geklickt, dann wird der Pfad der ersten Datei übernommen, die den Namen der Applikation von POV-Ray oder QuietPOV enthält. Das Eingabefeld für die Breite und Höhe der Grafik akzeptiert nur Zahlen und den Rücklauf. Die Eingabe ist auf vier Zeichen limitiert. Die Regler zur Positionierung der Kamera und der Lichtquelle können beliebig gestellt werden und führen zu keinem Fehler.
Im Kindfenster sind die Eingabefelder für Rekursionstiefe, Basiswinkel und Basisstärke darauf limitiert, nur Zahlen und den Rücklauf zu akzeptieren. Das Eingabefeld für die Rekursionstiefe ist auf zwei Zeichen begrenzt, beim Basiswinkel sind es drei und bei der Basisstärke zehn Zeichen. Wird auf den Button „Prüfen“ geklickt, werden die Produktionen des L-Systems auf mögliche Schwächen geprüft. Dazu zählt beispielsweise eine ungleiche Anzahl von öffnenden und schließenden sowie eckigen und geschweiften Klammern. Außerdem wird geprüft, ob die angegebenen Parameter im Produktionskopf verwendet werden oder Parameter, die nicht im Produktionskopf enthalten sind. Ebenfalls geprüft wird, ob für ein Symbol keine oder mehrere Produktionen existieren. Die Prüfung geht aber nicht so weit, Bedingungsausdrücke im Hinblick auf ihre Gültigkeit zu prüfen oder so weit zu kontrollieren, ob zu jedem Zeitpunkt eine Produktion angewendet werden kann. Deshalb wird auch eine Warnung ausgegeben, wenn für ein Symbol zwei Produktionen angegeben werden, die einen unterschiedlichen Bedingungsteil aufweisen.

6.2.3.2 Usability-Studie

Zu Beginn der Studie werden zwei Studenten der Informatik mit der Thematik konfrontiert. Aufgrund ihres Vorwissen konnte ihnen der Theorieteil in weniger als zwei Minuten vermittelt werden. Nach zehn Minuten waren die Studenten mit dem kompletten Programm vertraut. Zusätzlich dazu werden noch fünf weitere Personen für die Usability-Studie herangezogen, die über unterschiedliche Erfahrung und ein unterschiedliches Vorwissen verfügen. Das Ergebnis ist in Tabelle 6.9 zusammengefasst.

Alter	Beruf	3D-Wissen	Computer-Wissen	Theorie	Bild erzeugen	3D-Einstellungen	Produktionsbearbeitung
21	Kaufmann	Keine	Grundlagen	10 Min.	2 Min.	2 Min.	7 Min.
23	Tischler	Keine	Grundlagen	5 Min.	4 Min.	2 Min.	5 Min.
22	Azubi	Profi	Profi	5 Min.	1 Min.	1 Min.	3 Min.
23	Student	keine	Fortgeschritten	5 Min.	1 Min.	2 Min.	2 Min.
24	Abiturientin	Keine	Grundlagen	6 Min.	1 Min.	2 Min.	5 Min.

Tabelle 6.9: Ergebnis der Usability-Studie

Es kann gesagt werden, dass die Einarbeitungszeit bis zu zehn Minuten betragen kann, wenn das theoretische Vorwissen vorliegt. Dabei hat sich gezeigt, dass die Auswirkungen der Schieberegler durch den 3D-Kubus sehr gut vermittelt werden können.

7 Zusammenfassung und Ausblick zu dieser Arbeit

7.1 Zusammenfassung

Das zentrale Thema dieser Diplomarbeit sind die L-Systeme. Demzufolge werden sie im theoretischen Teil ausführlich behandelt. Dabei werden die wichtigsten L-Systeme und deren Erweiterungen für die Computergrafik vertieft betrachtet. Des Weiteren werden die besprochenen L-Systeme grob klassifiziert. Die wichtigsten L-Systeme werden zur Klassifikation auf einen Zeitstrahl gesetzt. Zusätzlich wird für die Computergrafik ein Entscheidungsautomat zu L-Systemen entwickelt. Eine weitere Klassifikation erfolgt über die Einteilung der L-Systeme in zwei Klassen sowie über die Eingliederung in die Chomskyhierarchie. Auf diese Weise entsteht ein Überblick über die Vielfalt und Mächtigkeit dieser Grammatik. Neben den theoretischen Aspekten der L-Systeme werden hierbei auch die Turtle-Interpretation dargestellt, mit denen es möglich ist, ganze Bäume und andere komplexe Objekte zu visualisieren. Nach einer kurzen Einführung in die Genetischen Algorithmen und in die Genetische Programmierung werden die Mutationsmöglichkeiten von L-Systemen vertieft betrachtet. Es zeigt sich, dass gerade die Genetischen Algorithmen (Evolutionäre Algorithmen) für L-Systeme von vielen Autoren bevorzugt verwendet werden und dass die Anzahl der Mutationsmöglichkeit sehr hoch ist.
Im Zusammenhang mit den Anwendungsmöglichkeiten wird auch vertieft auf die Evolutionsschleife und auf deren Anwendungsmöglichkeiten eingegangen. Aus theoretischer Sicht zeigt der Teil der Bilderzeugung einen hohen Freiheitsgrad, deshalb lässt er sich in viele Teilbereiche aufteilen. So kann man beispielsweise nicht nur die Grammatik mutieren lassen, sondern auch die Interpretationsvorschrift der Grammatik.
Da im Rahmen einer Diplomarbeit nicht alle Module implementiert werden können, wird auf bestehende Softwarepakete zurückgegriffen. Dabei ist die Wahl auf das Softwarepaket Lparser Kombination mit POV-Ray gefallen.
Beim Entwurf der Fassade zeigt sich, dass das Softwarepaket Lparser viele Schwächen und Fehler beinhaltet. Deshalb wird auf eine externe Weiterentwicklung von Lparsers zurückgegriffen, mit der sogar kontextsensitive und parametrisierte L-Systeme verarbeiten werden können. Trotzdem muss das Programm LViewer ersetzt werden. Dazu wird Lparser insoweit erweitert, dass er selbst eine möglichst gute Kameraposition berechnet. Außerdem muss die von LV2POVID erzeugt POV-Ray Datei korrigiert werden, weil die Kameraposition und die Position der Lichtquelle nicht richtig übernommen werden. Das Ergebnis ist eine stabile Fassade, die für den Evolutionsprozess eine gute Schnittstelle bietet.
Als Renderer wird POV-Ray mit einem Tool namens QuietPOV verwendet. Damit sind neben der Fassade noch vier weitere Programme notwendig. Bei der Implementierung der Fassade wird darauf Wert gelegt, dass der Quelltext möglichst wenige API-Funktionen von Windows verwendet, um in einer kurzen Zeit auf einem anderen Betriebssystem portiert werden zu können.
Um einen einfacheren Einstieg in die Modellierung mit L-Systemen zu bekommen und das eigentliche Arbeiten mit L-Systemen zu vereinfachen, wird eine graphische Oberfläche entwickelt. Für das Arbeiten mit der Oberfläche sind Kenntnisse zu L-Systemen notwendig. Neben den Editierfunktionen wird auch eine Prüfroutine für L-Systeme entwickelt. Außerdem wird mit Direct3D ein 3D-Kubus geschaffen. Mit der interaktiven Bedienung von Schiebereglern soll die Bedeutung der Faktoren für die relative Positionierung der Kamera und die Lichtquelle veranschaulicht werden.

7.2 Ausblick

Es folgen zum Schluss noch einige mögliche Erweiterungen der Software.
Seitens der aktuellen Konfiguration des Systems wäre es wünschenswert, das Programm Lparser performanter zu programmieren, weil die verwendete Version 5.1 das Beispiel „airhorse.ls“ nicht verarbeitet. Außerdem wäre eine gewisse Plausibilitätsprüfung von Vorteil, um Warnungen auszugeben, wenn zum Beispiel eine Regel nie verwendet wird.
Weiterhin interessant wäre eine Modularisierung der Software und die damit einhergehenden Freiheitsgrade.
So könnten verschiedenste L-Systeme verwendet werden, beispielsweise das selten vorkommende Table L-System. Man könnte sogar so weit gehen, mit Timed L-Systemen Animationen einzubinden. Einen zusätzlichen Freiheitsgrad erhält man durch den Modeller. Dieser würde ermöglichen, die Interpretationsvorschriften mutieren zu lassen. Dadurch kann aus einem Fahrbefehl schnell eine Richtungsänderung werden. Da der Modeller bevorzugt eine allgemeine 3D-Beschreibung erzeugen soll, kann das 3D-Modell mit einem Adapter in eine Beschreibung für verschiedenste Renderer transformiert werden. Dadurch wäre man nicht an ein bestimmtes Ray-Tracing Programm gebunden.
Seitens der graphischen Oberfläche könnte versucht werden, die Eingabe intuitiver zu designen. Das Ergebnis könnte auf eine X-Frog ähnliche Oberfläche hinauslaufen. X-Frog wird zum Modellieren von Pflanzen verwendet. Dabei werden die 3D-Modelle aus Komponenten modelliert. Nun könnte man den Nutzer für eine intuitivere Oberfläche Teile der Pflanze einzeln modellieren lassen und diese als Komponenten in das L-System aufnehmen. Diese Komponenten könnten interaktiv modelliert werden und dann in eine oder mehrere Produktionen transformiert werden. Der Nutzer könnte dann angeben, ob die Komponente mitwachsen soll oder „schreibgeschützt“ ihre Struktur beibehalten soll. Somit könnte ein Nutzer beim Erzeugen einer Pflanze ein Blatt erzeugen und als Komponente abspeichern. Da in der Pflanze des Nutzers keine anderen Blatttypen vorkommen, soll nur dieser eine Blatttyp verwendet werden und überall an der Pflanze verteilt werden. Das Blatt soll sich aber nicht weiter verändern aus der Sicht des Wortes, welches dahinter steht. Um jedoch nicht immer das gleiche Blatt zu haben, kann er dann über Parameter, die er dem Blatt übergibt, leichte Veränderungen hervorbringen.
Eine andere Erweiterung für die graphische Oberfläche ist das Managen von Projekten, so dass es zu jedem Projekt auch eine eigene CFG-Datei für die Fassade gibt. Alternativ könnte man die Schnittstelle von Lprocess erweitern und die CFG-Datei über die Kommandozeile angeben.

Weitreichendere Auswirkungen könnten durch Java und RMI-Methoden erreicht werden. So könnte man ein Java-Interface zur Fassade bauen, um ein Web-Interface zu kreieren oder sogar einen Web-Service zur Erzeugung von Bildern aus L-Systemen anzubieten. Man könnte auch die Fassade in Java implementieren und die externen Programme in ein separates Java-Programm mit einer RMI-Methode kapseln. Auf diese Weise müssten die externen Programme nicht mehr auf einem Computer liegen. Nun könnte die Fassade zu einem Steuerprogramm für die parallele Abarbeitung von einkommenden Jobs ausgebaut werden. Die externen Programme melden sich bei der Fassade an und bekommen Jobs zugeteilt. Dabei könnten sich Programme auch mehrfach bei der Fassade anmelden. So können zwei POV-Ray Programme auf verschiedenen Rechnern zur Verfügung stehen. Die Fassade würde in diesem Fall den Programmen ohne Auslastung die vorhandenen Jobs zuteilen.
Würde man die Bilderkennung von Pflanzen mit L-Systemen weiterführen, könnte man sich einen Roboter vorstellen, der Gartenaufgaben übernimmt. Er könnte dann beispielsweise anhand der erkannten Pflanze entsprechend düngen oder sie als Unkraut entfernen.

Anhang:

A Abbildungsverzeichnis

Abbildung 1.1: Phyllotaxis der Sonnenblume [De03] 1
Abbildung 1.2: Stellungen der Knospen [De03] 2
Abbildung 2.1: Transitionsdiagramm für den EA 7
Abbildung 2.2: Übergang vom Anfangszustand zum ersten Folgezustand 8
Abbildung 2.3: Einführung der Zellteilung und der entsprechende Folgezustand 9
Abbildung 2.4: Ableitungsbaum zum Beispiel 2.4 11
Abbildung 2.5-a: Ableitungsbaum für Beispiel 2.7 15
Abbildung 2.5-b: Ableitungsbaum für Beispiel 2.7 15
Abbildung 2.6: Ableitungsbaum vom Beispiel 2.8 18
Abbildung 2.7: Ableitungsbaum für das B0L-System 19
Abbildung 2.8: Zweidimensionaler fadenförmiger Organismus 20
Abbildung 2.9: Struktur einer Produktion eines parametrisierten 0L-Systems 21
Abbildung 2.10: Ableitungsbaum für das Beispiel 2.11 22
Abbildung 2.11: Signalverlauf in einem PIL-System 23
Abbildung 2.12: Die graphische Interpretation der ersten vier generierten Worte 29
Abbildung 2.13: Die Entwicklung einer Pflanze mit einem L-System 30
Abbildung 2.14: Turtle-Interpretation im dreidimensionalen Raum 31
Abbildung 2.15: Die 3D-Grafik zum Beispiel 2.16 34
Abbildung 2.16: Beispiel einer Membran 37
Abbildung 2.17: Ein vollständiges Eco-Grammar System [CPS95] 38
Abbildung 2.18: PAP eines GA [Sch03] 39
Abbildung 2.19: Crossover in GA 40
Abbildung 2.20: Mutation in GA 41
Abbildung 2.21: Automat als Datenstruktur 41
Abbildung 2.22: Lineare Struktur 42
Abbildung 2.23: Bäume als Datenstruktur 43
Abbildung 2.24: Crossover in GP – Lineare Befehlssequenz 44
Abbildung 2.25: Crossover in GP - Baum 44
Abbildung 2.26: Mutation beim Baum 45
Abbildung 2.27: Eine Produktion als Chromosom 47
Abbildung 2.28: Beispiele für Point Mutation 48
Abbildung 2.29: Inversion einer Produktion 49
Abbildung 2.30: Deletion bei einer Produktion 49
Abbildung 2.31: Beispiel für Duplication an einer Produktion 50
Abbildung 2.32: Mutation einer Produktion mit Translocation 50
Abbildung 2.33: Hinzufügen eines neuen Symbols 51
Abbildung 2.34: Der Mutator von L-System 52
Abbildung 2.35: Die nicht mutierte Pflanze 53
Abbildung 2.36: Eine Mutation der Blume 53
Abbildung 2.37: Eine zweite Mutation der Blume 54
Abbildung 2.38: Ein L-System als Baum dargestellt 55
Abbildung 2.39: Mutationsmöglichkeiten unter GP [Ja03] 55
Abbildung 2.40: Grobe Struktur des Systems 57
Abbildung 3.1: Baum, erzeugt von Lparser und gerendert mit POV-Ray 62
Abbildung 3.2: Beispiel für eine 3D-Grafik von L-System 63
Abbildung 3.3: 3D-Modell erzeugt von RayTraced Evolution 64

Abbildung 3.4: Eine Rosenblüte erstellt von LinSys3D 65
Abbildung 3.5: Eine Lilie erstellt mit L-studio 67
Abbildung 4.1: Module des Bilderzeugungsprozesses 72
Abbildung 4.2: Der theoretische, der praktische Ablauf sowie die verwendeten Programme 73
Abbildung 4.3: Strahlenverfolgung in einer Szene [ESK97] 78
Abbildung 4.4: Die gerenderte Szene 80
Abbildung 4.5: Ablaufdiagramm der Fassade 83
Abbildung 4.6: Beispielhafte Kameraposition für a) 3D-Szenen und b) 2D-Szenen 85
Abbildung 4.7: Bitbelegung der Variable `state` 88
Abbildung 5.1: Das Hauptfenster von „Visual L“ 93
Abbildung 5.2: Das Kindfenster von „Visual L“ 94
Abbildung 5.3: Das Dialog-Fenster zum Editieren der CFG-Datei 96
Abbildung 5.4: Die 3D-Ansicht mit Direct3D 97
Abbildung 6.1: Tortendiagramm der Zeitaufteilung 101

B Tabellenverzeichnis

Tabelle 2.1: Übergangsmatrix für den EA 7
Tabelle 2.2: Übergangsmatrix für den EA 8
Tabelle 2.3: Übersichtstabelle der besprochenen L-Systeme 25
Tabelle 3.1-a: Übersicht über die Software zu L-Systemen 68
Tabelle 3.1-b: Übersicht über die Software zu L-Systemen 68
Tabelle 4.1: Unterschiede zwischen Version 4 und 5 76
Tabelle 4.2: Kommandozeilenargumente der Fassade 80
Tabelle 4.3: Alle Werte der LPROCESS.CFG 81
Tabelle 4.4: Module der Fassade 87
Tabelle 6.1: Zeit zum Konvertieren von 117 Dateien 99
Tabelle 6.2: Zeit für die Verarbeitung von 101 Dateien vom Lparser in der Debug-Version . 99
Tabelle 6.3: Zeit für die Verarbeitung von 101 Dateien vom Lparser in der Release-Version 99
Tabelle 6.4: Zeit für das Transformieren von 95 Dateien 100
Tabelle 6.5: Zeit für das Erzeugen von 95 Grafiken ohne Anti-Aliasing 100
Tabelle 6.6: Zeit für das Erzeugen von 95 Grafiken mit Anti-Aliasing 100
Tabelle 6.7: Zeit für die Verarbeitung von 95 Dateien mit Lprocess 100
Tabelle 6.8: Ergebnis der Belastungen der einzelnen Komponenten 101
Tabelle 6.9: Ergebnis der Usability-Studie 104

C Literaturverzeichnis

[Ab86] Lois A. **Abbott**. *Investigations into Drosophila Wing Development – Result from a Lindenmayer Model*. *The Book of L, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 1 - 11, Springer-Verlag, 1986.

[Ba98] Helmut **Balzert**. *Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung*. Spektrum Akademischer Verlag, 1998.

[Ba99] Heide **Balzert**. *Lehrbuch der Objektmodellierung: Analyse und Entwurf*. Spektrum Akademischer Verlag, 1999.

[Ba00] Helmut **Balzert**. *Lehrbuch der Software-Technik: Software-Entwicklung*. Spektrum Akademischer Verlag, 2000.

[Ban03] Wolfgang **Banzhaf**. *Einführung in das Genetische Programmieren*. *Vorlesungsskript*, Universität Dortmung Fachbereich Informatik, 2003. http://ls11-www.cs.uni-dortmund.de/people/banzhaf/GP/GP-Vorlesung.pdf (Stand: 06/2004).

[Be01] Martin **Bellardi**. *Lindenmayer-Systeme*. Referate zum Seminar: *Zufall+Rauschen*, 2001. http://bellardi.de/pdf/L-Systeme.pdf (Stand: 06/2004).

[BNKF98] Wolfgang **Banzhaf**, Peter **Nordin**, Robert E. **Keller** und Frank D. **Francone**. *Genetic Programming: An Introduction*. dpunkt Verlag, 1998.

[Bo04] Ingo **Boersch**. *Evolution*. *Vorlesungsskript zu Wissensverarbeitung*, Fachhochschule Brandenburg Fachbereich Informatik und Medien, 2004. http://ots.fh-brandenburg.de/mod/ams/data/wva/unterlagen/evo.pdf (Stand: 06/2004).

[BPFGK03] Frédéric **Boudon**, Przemyslaw **Prusinkiewicz**, Pavol **Federl**, Christophe **Godin** und Radoslaw **Kardowski**. *Interactive design of bonsai tree models*. *Proceedings of Eurographics 2003, Hrsg. P. Brunet und D. Fellner*, Band 22, 2003. http://www.cpsc.ucalgary.ca/Research/bmv/papers/index.html (Stand: 06/2004).

[Bü03] Frank **Büttgen**. *Prozeduraler Städtebau*. *Seminararbeit*, Rheinisch-Westfälische Technische Hochschule Aachen, 2003.

[CJ92] T. W. **Chien** und Helmut **Jürgensen**. *Parameterized L Systems for Modelling: Potential and Limitations*. *Lindenmayer Systems, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 213 - 229, Springer-Verlag, 1992.

[CK86] Karl **Culik** und Juhani. **Karhumäki**. *A New Proof for the D0L Sequence Equivalence Problem and Its Implications*. *The Book of L, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 63 - 74, Springer-Verlag, 1986.

[CPS95] Erzsebet **Csuhaj**-Varju, Gheorghe **Păun** und Arto **Salomaa**. *Conditional Tabled Eco-Grammar Systems versus (E)T0L-Systems*. Journal of Universal Computer Science, Band: 1, S. 252 - 268, Springer-Verlag, 1995. http://www.jucs.org/jucs_1_5/conditional_tabled_eco_grammar/paper.pdf (Stand: 06/2004).

[Cs02] Judit **Csima**. *Investigations on Simple Eco-Grammar Systems*. *Ph. D. Dissertation*, Eötvös Loránd University, 2002. http://sziami.cs.bme.hu/~csima/phd1.ps (Stand: 06/2004).

[DB03] Roger Luke **DuBois**. *Applications of Generative String-Substitution Systems in Computer Music*. *Ph. D. Dissertation*, University of Columbia, 2003. http://www.music.columbia.edu/~luke/dissertation/dissertation.pdf (Stand: 06/2004).

[De03] Oliver **Deussen**. *Computergenerierte Pflanzen*. Springer-Verlag, 2003.

[DK00] Walter **Doberenz** und Thomas **Kowalski**. *Borland Delphi 5: Grundlagen und Profiwissen*. Carl Hanser Verlag, 2000.

[DS98] Stephen R. **Davis** und Richard J. **Simon**. *Win98 Programmierung für Dummies*. MITP-Verlag GmbH, 1998.

[Du00] Robert L. **Dunlop**. *DirectX 7 Programmierung*. Markt+Technik Verlag, 2000.

[Eb02] Michael **Ebner**. *Delphi 6 nachschlagen und verstehen*. Addison-Wesley Verlag, 2002.

[EG01] Wolfgang F. **Engel** und Amir **Geva**. *Direct 3D Spieleprogrammierung*. SYBEX-Verlag GmbH, 2001.

[El00] Frank **Eller**. *workshop Delphi 5*. Addison-Wesley Verlag, 2000.

[El01] Frank **Eller**. *workshop Delphi 6*. Addison-Wesley Verlag, 2001.

[ESK96] José **Encarnação**, Wolfgang **Straßer** und Reinhard **Klein**. *Graphische Datenverarbeitung 1*. R. Oldenbourg Verlag, 1996.

[ESK97] José **Encarnação**, Wolfgang **Straßer** und Reinhard **Klein**. *Graphische Datenverarbeitung 2*. R. Oldenbourg Verlag, 1997.

[FMF04] Bettina **Fiege**, Ute **Manthey** und Gabriele **Frank**. *Persönliche E-Mail Korrespondenz (auf CD-ROM der Diplomarbeit enthalten)*. Auszüge auf: http://www.uni-jena.de/data/unijena_/faculties/minet/casio/Lindenmayer2.html (Stand: 06/2004).

[Fo00] Charles **Fox**. *Genetic Hierarchical Music Structures*. *Ph. D. Dissertation*, Clare College Cambridge, 2000. http://www.robots.ox.ac.uk/~charles/mg/dissertation.doc (Stand: 06/2004).

[Gä96] Stefan **Gärtner**. *Partitions-limierte Lindenmayer-Systeme*. Shaker Verlag, 1996.

[GFMP01] Tobias **Gross**, Nils **Faltin**, Roman **Mülchen** und Michael **Plath**. *Computergrafik Interaktiv - Grafiti*. Computer Graphics & Software Ergonomie Carl von Ossietzky Universität Oldenburg, 2001. http://olli.informatik.uni-oldenburg.de/Grafiti3/grafiti/flow1/page1.html (Stand: 06/2004).

[GHJV96] Erich **Gamma**, Richard **Helm**, Ralph **Johnson** und John **Vlissides**. *Entwurfsmuster*. Addison Wesley Verlag, 1996.

[Gw03] Brad **Goodwin**. *The Wonderful World of Birds and Feathers*. *CS 563 Advance Topics in Computer Graphics*, Worcester Polytechnic Institute, 2003. http://www.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/bradg/feathers/featherpaper_files/feathers.ppt (Stand: 06/2004).

[Gö99] Wilhelm **Göhler**. *Formelsammlung: Höhere Mathematik*. Verlag Harri Deutsch, 1999.

[GR92] Narendra S. **Goel** und Ivan **Rozehnal**. *A High-Level Language for L-systems and Its Applications*. *Lindenmayer Systems Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 231 - 251,Springer-Verlag, 1992.

[HLP01] Gregory S. **Hornby**, Hod **Lipson** und Jordan B. **Pollack**. *Evolution of Generative Design Systems for Modular Physical Robots*. IEEE International Conference on Robotics and Automation, 2001. http://www.demo.cs.brandeis.edu/papers/author.html#hornby (Stand: 06/2004).

[Ho03] Gregory S. **Hornby**. *Generative Representation for Evolutionary Design Automation*. *Ph. D. Dissertation*, Brandeis University Dept. of Computer Science, 2003. http://www.demo.cs.brandeis.edu/papers/author.html#hornby (Stand: 06/2004).

[Hol01] Markus **Holenstein**. *Aesthetic Selection*. *Diplomarbeit*, Universität Zürich, 2001.

[Hon01] Juha **Honkala**. *Three Variants of the DT0L Sequence Equivalence Problem*. Journal of Universal Computer Science, Band: 7, S. 886 – 892, Springer-Verlag, 2001. http://www.jucs.org/jucs_7_10/three_variants_of_the/paper.pdf (Stand: 06/2004).

[HP01a] Gregory S. **Hornby** und Jordan B. **Pollack**. *Evolving L-Systems To Generate Virtual Creatures*. Computers and Graphics, 25:6, S. 1041 - 1048, 2001. http://www.demo.cs.brandeis.edu/papers/author.html#hornby (Stand: 06/2004).

[HP01b] Gregory S. **Hornby** und Jordan B. **Pollack**. *The Advantages of Generative Grammatical Encodings for Physical Design*. Congress on Evolutionary Computation, 2001. http://www.demo.cs.brandeis.edu/papers/author.html#hornby (Stand: 06/2004).

[HP01c] Gregory S. **Hornby** und Jordan B. **Pollack**. *Body-Brain Co-evolution Using L-systems as a Generative Encoding*. Genetic and Evolutionary Computation Conference, 2001.
http://www.demo.cs.brandeis.edu/papers/author.html#hornby (Stand: 06/2004).

[HP02] Gregory S. **Hornby** und Jordan B. **Pollack**. *Creating High-Level Components with a Generative Representation for Body-Brain Evolution*. Artifical Life, 8:3, 2002.
http://www.demo.cs.brandeis.edu/papers/author.html#hornby (Stand: 06/2004).

[HR75] Gabor T. **Herman** und Grzegorz **Rozenberg**. *Developmental Systems and Languages*. North-Holland Publishing Company und American Elsevier Publishing Company, 1975.

[HU00] John E. **Hopcroft** und Jeffrey D. **Ullman**. *Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie*. Oldenbourg Wissenschaftsverlag GmbH, 2000.

[Ja95] Christian **Jacob**. *Genetic L-System Programming: Breeding and Evolving Artificial Flowers with Mathematica*. *First International Mathematica Symposium*, Computational Mechanics Pub., 1995.
http://pages.cpsc.ucalgary.ca/~jacob/IMS-ArtFlowers.pdf (Stand: 05/2004).

[Ja97] Christian **Jacob**. *Principia Evolvica : Simulierte Evolution mit Mathematiker*. dpunkt Verlag, 1997.

[Ja03] Christian **Jacob**. *Artificial Intelligence through Evolution*. *Vorlesungsskript*, University of Calgary, 2003.
http://pages.cpsc.ucalgary.ca/~jacob/Courses/Fall2003/CPSC533/Slides/09-GP.pdf (Stand: 06/2004).

[JAH01] Ron **Jeffries**, Ann **Anderson** und Chet **Hendrickson**. *Extrem Programming installed*. Addison-Wesley Verlag, 2001.

[JM86] Helmut **Jürgensen** und David E. **Matthews**. *Stochastic 0L Systems and Formal Power Series*. *The Book of L, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 167 - 177, Springer-Verlag, 1986.

[Ke86] Alica **Kelemenová**. *Complexity of L-Systems*. *The Book of L, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 179 - 191, Springer-Verlag, 1986.

[Ko99] Andreas **Kosch**. *COM/DCOM für Delphi*. Software & Support Verlag, 1999.

[KPM92] Kamala **Krithivasan**, M. V. Nagendra **Prasad** und Meena **Mahajan**. *„Forgetful" L Systems*. *Lindenmayer Systems, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 419 - 436, Springer-Verlag, 1992.

[KR90] Brian W. **Kernighan** und Dennis M. **Ritchie**. *Programmieren in C*. Carl Hanser Verlag, 1990.

[KR92] Alica **Kelemenová** und Miriam **Removčíková**. *A0L- and CFG- Size of Languages*. *Lindenmayer Systems, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 177 - 182, Springer-Verlag, 1992.

[KTV99a] Gabriella **Kókai**, Zoltán **Tóth** und Róbert **Ványi**. *Modelling Blood Vessels of the Eye with Parametric L-Systems Using Evolutionary Algorithms*. *Proceedings Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making*, S. 433 - 443, Springer-Verlag, 1999. http://www2.informatik.uni-erlangen.de/download/Papers/ (Stand: 06/2004).

[KTV99b] Gabriella **Kókai**, Zoltán **Tóth** und Róbert **Ványi**. *Parametric L-System Description of the Retina with Combined Evolutionary Operators*. *Proceedings GECCO, Genetic and Evolutionary Computation Conference*, Band: 2, S. 1588 - 1596, 1999. http://www2.informatik.uni-erlangen.de/download/Papers/ (Stand: 06/2004).

[Ku99] Winfried **Kurth**. *Die Simulation der Baumarchitektur mit Wachstumsgrammatiken*. Wissenschaftlicher Verlag Berlin, 1999.

[Kud86] Manfred **Kudlek**. *Languages Defined By Indian Parallel Systems*. *The Book of L, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 233 - 243, Springer-Verlag, 1986.

[LA98] Richard C. **Leinecker** und Tom **Archer**. *Die Windows 98 Programmier-Bibel*. ITP-Verlag GmbH, 1998.

[Li68a] Aristid **Lindenmayer**. *Mathematical Models for Cellular Interactions in Develeopment. I. Filaments with One-sided Inputs*. J. Theoret. Biol., Heft 18 / 1968, S. 280 - 299, 1968.

[Li68b] Aristid **Lindenmayer**. *Mathematical Models for Cellular Interactions in Develeopment. II. Simple and Branching Filaments with Two-sided Inputs*. J. Theoret. Biol., Heft 18 / 1968, S. 300 - 315, 1968.

[Lo00] Dirk **Louis**. *Delphi 5 New Reference*. Markt + Technik Verlag, 2000.

[LS92] Klaus-Jörn **Lange** und Michael **Schudy**. *The Complexity of the Emptiness Problem for E0L Systems*. *Lindenmayer Systems, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 167 - 175, Springer-Verlag, 1992.

[LT86] J. van **Leeuwen** und R. B. **Tan**. *Computer Networks with Compact Routing Tables*. *The Book of L, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 259 - 273, Springer-Verlag, 1986.

[Mi93] **Microsoft** Corporation. *Microsoft MS-DOS 6.2 Benutzerhandbuch*. Microsoft Corporation, 1993.

[Mi99] Zbigniew **Michalewicz**. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1999.

[Mo96] Kenrick **Mock**. *Introduction to Artificial Life and Genetic Algorithms*. *Vorlesungsskript*, University of Alaska, 1996. http://www.math.uaa.alaska.edu/~afkjm/cs405/handouts/genetic.ppt (Stand: 05/2004).

[NLA86] A. **Nakamura**, Aristid **Lindenmayer** und K. **Aizawa**. *Some Systems for Map Generation*. *The Book of L Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 323 - 332, Springer-Verlag, 1986.

[OW02] Thomas **Ottmann** und Peter **Widmayer**. *Algorithmen und Datenstrukturen*. Spektrum Akademischer Verlag, 2002.

[Pa92] Gheorghe **Păun**. *Parallel Communicating Systems of L Systems*. *Lindenmayer Systems Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 405 - 417, Springer-Verlag, 1992.

[Pa98] Gheorghe **Păun**. *Computing with Membranes*. 1998. http://psystems.disco.unimib.it/ (Stand: 06/2004).

[Pa00] Gheorghe **Păun**. *From Cells to Computers: Computing with Membranes (P Systems)*. Präsentiert auf dem *Workshop on Grammar Systems*, 2000. http://psystems.disco.unimib.it/ (Stand: 06/2004).

[PHHM95] Przemyslaw **Prusinkiewicz**, Mark **Hammel**, Jim **Hanan** und Radomìr **Měch**. *The Artificial Life of Plants*. *Artificial life for graphics, animation, and virtual reality*, Band 7 von SIGGRAPH ’95, S. 1 – 38, ACM Press, 1995. http://www.cpsc.ucalgary.ca/Research/bmv/papers/index.html (Stand: 06/2004).

[PHHM96a] Przemyslaw **Prusinkiewicz**, Mark **Hammel**, Jim **Hanan** und Radomìr **Měch**. *Visual models of plant development*. *Handbook of formal languages, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, Springer-Verlag, 1996. http://www.cpsc.ucalgary.ca/Research/bmv/papers/index.html (Stand: 06/2004).

[PHHM96b] Przemyslaw **Prusinkiewicz**, Mark **Hammel**, Jim **Hanan** und Radomìr **Měch**. *L-Systems: From The Theory to Visual Models of Plants*. *Proceedings of the 2nd CSIRO Symposium on Computational Challanges in Life Sciences*, CSIRO Publishing, 1996. http://www.cpsc.ucalgary.ca/Research/bmv/papers/index.html (Stand: 06/2004).

[PHM] Przemyslaw **Prusinkiewicz**, Jim **Hanan** und Radomìr **Měch**. *Extensions to the graphical interpretation of L-systems based on turtle geometry*. University of Calgary Dept. of Computer Science. http://www.cpsc.ucalgary.ca/Research/bmv/lstudio/graph.pdf (Stand: 06/2004).

[PJM94] Przemyslaw **Prusinkiewicz**, Mark **James** und Radomìr **Měch**. *Synthetic Topiary*. Proceedings of SIGGRAPH 94, S. 351- 358, 1994. http://www.cpsc.ucalgary.ca/Research/bmv/papers/index.html (Stand: 06/2004).

[PL90] Przemyslaw **Prusinkiewcz** und Aristid **Lindenmayer**. *The Algorithmic Beauty of Plants*. Springer-Verlag, 1990.

[PP99] Ulla Kirch-**Prinz** und Peter **Prinz**. *C für PCs*. MITP-Verlag GmbH, 1999.

[Pr86] Przemyslaw **Prusinkiewicz**. *Graphical Applications of L-Systems*. Proceedings of SIGGRAPH 1986, S. 247 - 253, 1986.

[Pr93] Przemyslaw **Prusinkiewicz**. *Modeling and Visualization of Biological Structures*. Proceedings of Graphics Interface 1993, S. 128 - 137, 1993. http://www.cpsc.ucalgary.ca/Research/bmv/papers/index.html (Stand: 06/2004).

[PR01] Gheorghe **Păun** und Grzegorz **Rozenberg**. *A Guide to Membrane Computing*. 2001. http://psystems.disco.unimib.it/ (Stand: 06/2004).

[Qu02] Uwe **Quasthoff**. *Genetische Programmierung Teil 1 und Teil 2. Vorlesungsskript*, Universität Leipzig Institut für Informatik, 2002. http://wortschatz.informatik.uni-leipzig.de/asv/lehre/ss02/QGenProgrGesamt080702.pdf (Stand: 06/2004).

[Ra04] Jürgen **Rathlev**. *Direct3D mit Delphi unter DirectX 8*. Delphi-Source.de, 2004. http://www.delphi-source.de/downloads/dcount.php?cat=t&id=tutorials/direct3d/direct3d.zip (Stand: 06/2004).

[Ras95] Mathias **Rasch**. *Grafikprogrammierung mit Windows 95*. DATA BECKER GmbH & Co. KG, 1995.

[Ri97] Jeffrey **Richter**. *Microsoft Windows Programmierung für Experten*. Microsoft Press Deutschland, 1997.

[Ro04] Grzegorz **Rozenberg**. *Grzegorz Rozenberg*. Grzegorz Rozenberg, 2004. http://www.wi.leidenuniv.nl/home/rozenber/ (Stand: 06/2004).

[Rou01] Christian **Rousselle**. *Spieleprogrammierung mit DirectX und Visual C++*. Markt+Technik Verlag, 2001.

[RS86] Grzegorz **Rozenberg** und Arto **Salomaa**. *The Book of L*. Springer-Verlag, 1986.

[Ru86] Keijo **Ruohonen**. *Equivalence Problems for Regular Sets of Word Morphisms*. *The Book of L, Hrsg. Grzegorz Rozenberg und Arto Salomaa*, S. 393 - 401, Springer-Verlag, 1986.

[Sc02] Thorsten **Schnier**. *Lindenmayer Systems Part II: Nature Inspired Design*. University of Birmingham School of Computer Science, 2002. http://www.cs.bham.ac.uk/~xin/courses/design/11-l-systems-1-4up.pdf (Stand: 06/2004).

[Sch03] Tino **Schonert**. *Einsatz evolutionärer Algorithmen zur Optimierung der Tourenplanung eines Wachschutzunternehmens*. *Diplomarbeit*, Fachhochschule Brandenburg Fachbereich Information und Medien, 2003. http://ots.fh-brandenburg.de/mod/diplom/data/da_tino_schonert.pdf (Stand: 06/2004).

[SD01] Wolfgang **Scholl** und Rainer **Drews**. *Handbuch Mathematik*. Orbis Verlag, 2001.

[SDU99] Victor **Sirotin**, Victor **Debeloff** und Yuri **Urri**. Dire*ctX-Programmierung mit Visual C++*. Addison Wesley Longman Verlag, 1999.

[Se92] Robert **Sedgewick**. *Algorithmen*. Addison Wesley Longman Verlag, 1992.

[SGT99] Wolfgang **Schäfer**, Kurt **Georgi** und Gisela **Trippler**. *Mathematik-Vorkurs*. B. G. Teubner, 1999.

[Si97] ACM **SIGGRAPH**. *1997 ACM SIGGRAPH Awards: Computer Graphics Achievement Award Przemyslaw Prusinkiewcz*. ACM SIGGRAPH, 1997. http://www.siggraph.org/awards/1997/AchievementAward.html (Stand: 06/2004).

[SH03] Arne **Schäpers** und Rudolf **Huttary**. *Daniel Düsentrieb: C#, Java, C++ und Delphi im Effizienztest, Teil 2*. c't, Heft: 21/2003, ab S. 222, Heise Zeitschriften Verlag, 2003.

[To99] Hrsg. **Toolbox** Redaktion. *Visual C++ 6*. C&L Computer und Literaturverlag, 1999.

[Wa01] Johannes **Waldmann**. *L-Systeme*. *Vorlesungsskript*, Universität Leipzig Institut für Informatik, 2001. http://www.informatik.uni-leipzig.de/~joe/edu/ss01/l/l-pub.ps (Stand: 06/2004).

[Wä99] Dietmar **Wätjen**. *Automatentheorie und Formale Sprachen: Vorlesungsskript*. Skript zur Vorlesung *Automatentheorie und Formale Sprachen* an der Technischen Universität Braunschweig, 1999.

[Wi95] Gerhard **Willms**. *Das C Grundlagen Buch*. DATA BECKER GmbH & Co. KG, 1995.

[Za01] Mair **Zamir**. *Arterial Branching within the Confines of Fractal L-System Formalism*. J. Gen. Physiol., Band: 118, S. 267 - 275, The Rockefeller University Press, 2001. http://www.jpg.org/cgi/content/full/118/3/267 (Stand: 05/2004).

D Zusammenfassung der wichtigsten Literaturquellen

Zur Theorie von L-Systemen:
[HR75] [PL90]

Zur graphischen Interpretation von L-Systemen:
[PL90]

Zu Genetischen Algorithmen und Genetischer Programmierung:
[Mi99] [Ja97] [BNKF98] [Bo04]

Zur Mutation von L-Systemen:
[Ja03] [Ja97] [Hol01]

Zur Programmentwicklung mit C:
[KR90] [PP99]

Zur Programmentwicklung mit Delphi:
[DK00] [Lo00] [El00] [El01]

Zur Windows-Programmierung:
[Ri97]

Zur DirectX-Programmierung:
[Ra04] [Du00]

E Softwarequellen

[[DX8]] **DirectX 8**-Units für Delphi, Kapslung der DirectX-API für Delphi, Open Source (die Kapslung, nicht DirectX selbst).
http://www.delphi-source.de/downloads/dcount.php?cat=t&id=tutorials/direct3d/direct3d8.zip (Stand: 06/2004).

[[Fra]] **Fractint** Version 20.0 für MS-DOS, läuft unter Windows in einer DOS-Box, erzeugt Bilder aus Fraktalen, Open Source.
http://spanky.triumf.ca/pub/fractals/programs/ibmpc/frain200.zip (Stand: 06/2004).

[[Gra]] **GraphicEx** image library Version 9.9 für Delphi ab Version 4.0, Erweiterung der Grafikbibliothek von Delphi um weitere Dateiformate, Open Source.
http://scripts.soft-gems.net/download.php?ID=13 (Stand: 06/2004).

[[Gro]] **GROGRA** Version 3.3 für Windows, Simulation von Wachstumsgrammatiken, Freeware.
http://www.grogra.de/ (Stand: 06/2004). Erwerb der Software nur über den direkten Kontakt mit Prof. Dr. Winfried Kurth.

[[Imo]] **Imogene** Version 1.0 für Windows, erzeugt durch künstliche Evolution Grafiken, die von einem Nutzer bewertet werden, Shareware.
ftp://bells.cs.ucl.ac.uk/genetic/ftp.io.com/code/imogenes.zip (Stand: 06/2004).

[[JVCL]] **JEDI Visual Component Library** Version 2.10 für Delphi ab Version 5.0, Sammelsurium von verschiedensten Komponenten, Open Source.
http://prdownloads.sourceforge.net/jvcl/JCL%2BJVCL210FullInstall.zip?use_mirror=switch (Stand: 06/2004). Wichtiges Fix für die Version 2.10
http://prdownloads.sourceforge.net/jvcl/JVCL210FIX030313.zip?use_mirror=switch (Stand: 06/2004).

[[L32]] **LSys32** für Windows, grafische Oberfläche für den Lparser Version 4.0, Freeware.
http://chris.lichti.org/Lab/LSys32/lsys.zip (Stand: 06/2004).

[[L3D]] **LinSys3d** Version 1.2 für Windows, erzeugt 3D-Szenen aus L-Systemen, Freeware.
http://web.tiscali.it/esuli/LinSys3d/LS3dSetup.exe (Stand: 06/2004).

[[Lap]] **L-Systems Application**, Java Applet, erzeugt 2D-Grafiken aus L-Systemen.
http://www.cpsc.ucalgary.ca/Research/bmv/java/LSystems/LSys.html (Stand: 06/2004).

[[LP4]] **Lparser Version 4.0** für MS-DOS, läuft unter Windows in einer DOS-Box, erzeugt 3D-Szenen aus L-Systemen, Freeware.
http://home.wanadoo.nl/laurens.lapre/lparser4.zip (Stand: 06/2004).

[[LP5]] **Lparser Version 5.1** für Windows, erzeugt 3D-Szenen aus L-Systemen, Open Source.
http://vortex.bd.psu.edu/~kopp/Lparser.zip (Stand: 06/2004).

[[LS4]] **L-System 4.0.1** für Windows, erzeugt 3D-Szenen aus L-Systemen, Freeware.
http://www.geocities.com/tperz/l4setup.zip (Stand: 06/2004).

[[LSB]] **LS-SketchBook** Version 0.1 für Windows, erzeugt 3D-Szenen aus L-Systemen, Freeware.
http://coco.ccu.uniovi.es/malva/sketchbook/lssketchbook/download/LS%20SketchBook%2001b.zip (Stand: 06/2004).

[[LSt]] **L-studio** Version 3.1 für Windows, erzeugt 3D-Szenen aus L-Systemen, Kommerzielle Software.
http://www.cpsc.ucalgary.ca/Research/bmv/lstudio/lstudio-demo.zip
(Stand: 06/2004).

[[LWo]] **Lworld** Version 7.0 für Windows, erzeugt Animationen aus L-Systemen, Freeware.
http://www.ifi.unizh.ch/~noser/Lworld/LworldWinPCv7.zip (Stand: 06/2004).

[[Plan]] **PlanD** für Windows, Entwicklungssystem für L-Systeme, Freeware.
http://www.imn.htwk-leipzig.de/~sdietze1/planD.zip (Stand: 06/2004).

[[Pov]] **POV-Ray** Version 3.5 für Windows, Ray Tracer, Freeware.
http://www.povray.org/redirect/www.povray.org/ftp/pub/povray/Official/Windows/povwin35.exe (Stand: 06/2004).

[[Qpov]] **QuietPOV** Version 1.1 für Windows, GUI Extension für POV-Ray, Freeware.
http://www.throwable.com/dreampeach/download/QuietPOV/QuietPOV-install.exe (Stand: 06/2004).

[[RTE]] **RayTraced Evolution** Version 1.0b3 für Windows, erzeugt 3D-Szenen aus L-Systemen, Freeware.
http://www.rz.tu-ilmenau.de/~juhu/GX/RTEvol/win/rte.zip (Stand: 06/2004).

F Bedienungsanleitung für Visual L

Voraussetzung für das Benutzen der Software ist eine Grafikkarte die, DirectX 8 unterstützt, sowie eine installierte Version von POV-Ray 3.5 und QuietPOV.

Für den ersten schnellen Einstieg in die Software öffnen Sie eine bestehende LS-Datei, indem Sie in der Symbolleiste auf „Öffnen“ klicken oder über das Menü „Datei“ auf „Öffnen“ gehen. Dort suchen Sie sich eine vorhandene LS-Datei aus und öffnen diese.
Nach dem Öffnen erscheint ein neues Fenster, siehe Abbildung 5.2. Auf der rechten Seite des Fensters befinden sich mehrere Buttons. Einer besitzt die Aufschrift „Bild erzeugen“, auf diesen klicken Sie, um die 3D-Grafik für das L-System zu erhalten.
Die Bilderzeugung kann unter Umständen länger dauern. Wenn das Bild erzeugt wurde, öffnet sich ein Fenster, in dem die 3D-Grafik enthalten ist. Parallel dazu liegt die 3D-Grafik im Verzeichnis von „Visual L“. Durch einen Klick auf das Bild schließt sich das Fenster wieder.
Wenn Ihnen das Bild zu klein ist, können Sie auf den Reiter „Einstellungen“ des Fensters klicken, in dem das L-System geöffnet ist, und dort die Höhe und Breite der Grafik in Pixel angeben. Außerdem können Sie angeben, ob das Targa-Format zur Speicherung der Grafik verwendet werden soll und ob beim Erstellen der 3D-Grafik die Anti-Aliasing Funktion aktiviert werden soll. Das Anti-Aliasing entfernt unschöne Treppeneffekte in der 3D-Grafik, durch das Aktivieren dieser Option kann die Bilderzeugung jedoch bis zu 40 % länger dauern.
Nicht immer entsprechen die voreingestellte Kamera- und Lichtposition den eigentlichen Wünschen. Um eine Umpositionierung vorzunehmen, kann auf den Reiter „Kamera und Licht“ geklickt werden. Auf dieser Seite befinden sich einige Schieberegler, mit denen sich die Werte für die Kamera- und Lichtposition verändern lassen. Damit die Auswirkungen besser verständlich sind, kann ein 3D-Kubus zugeschaltet werden, indem auf „3D-Ansicht“ geklickt wird. Der 3D-Kubus ist nur exemplarisch und soll keine Vorschau der späteren Grafik darstellen. Er ist lediglich eine Vorschau über die Auswirkungen der Schieberegler und der Umpositionierung der Licht- und Kameraposition. Wenn das Fenster mit dem 3D-Kubus geöffnet wird, können an den Schiebereglern andere Werte eingestellt werden, der 3D-Kubus aktualisiert sich von selbst mit den neuen Werten. Anzumerken ist, dass die Lichtposition relativ zur Kameraposition ist, d. h. wenn die Kamera verschoben wird, verschiebt sich auch automatisch das Licht.
All diese Einstellungen können auch über das Menü „Einstellungen“ und „CFG-Datei“ vorgenommen werden. Zusätzlich zu den schon bekannten Einstellungen kann der Pfad von POV-Ray und QuietPOV eingestellt bzw. gesucht werden. Diese Einstellungen werden in einer Textdatei namens „LPROCESS.CFG“ festgehalten. Sollte die Datei gelöscht sein oder einen ungültigen Wert enthalten, dann meldet sich „Visual L“ und erzeugt eine neue CFG-Datei oder benutzt Standardwerte.
Wenn Sie auf den Button „Bild erzeugen“ klicken, wird im Hintergrund eine andere Applikation namens „Lprocess“ gestartet, diese wiederum startet weitere Programme, um das Bild zu erzeugen. „Lprocess“ selbst und die Programme „Lparser“ und „LV2POVID“ befinden sich im Verzeichnis von „Visual L“. Alle drei Programme und die im Verzeichnis befindlichen CFG-Dateien werden benötigt, um das Bild zu generieren.

Nun soll ein eigenes L-System definiert werden. Dazu klicken Sie in der Symbolleiste auf „Neu“ oder auf den Menüpunkt „Datei“ und dann „Neu“. Dieses Mal erscheint ein leeres Fenster, in dem noch nichts eingetragen ist.
Auf dem Reiter „L-System“ befinden sich zwei größere Bereiche. Der eine ist mit „Startwerte“ betitelt und der andere mit „Produktionen“. Im Bereich „Startwerte“ können Sie die Rekursionstiefe (wie oft das Wort abgeleitet werden soll), den Start-/Basiswinkel (für die Turtle-Interpretation), die Start-/Basisstärke (wie dick/stark die 3D-Zylinder sein sollen) und das Startaxiom/-wort angeben.
Die Felder können Sie mit folgenden Werten füllen: die Rekursionstiefe beträgt zwei, der Basiswinkel beträgt 90, die Basisstärke ist 100 und das Startaxiom ist „F-F-F-F“.
Im unteren Bereich können die einzelnen Produktionen/Regeln für das L-System definiert werden. Die Eingabemaske für eine Produktion besteht aus 5 Feldern. Das erste ist der linke Kontext der Produktion, gefolgt vom Produktionskopf. Das dritte Feld ist der rechte Kontext und das vierte die Bedingung, unter der die Produktion ausgeführt werden soll. Das letzte Feld ist der Produktionskörper, der beim Ableiten für das Symbol im Produktionskopf ersetzt wird. Sollten der linke oder rechte Kontext oder der Bedingungsteil nicht benötigt werden, dann kann ein „*“ eingesetzt werden. Wurde die Produktion definiert, kann auf „Hinzufügen“ geklickt werden. Sollte die Produktion fehlerhaft sein, erscheint eine Meldung. Als kleine Hilfe für die Erstellung der Produktionen können zwei Tabellen eingeblendet werden. Die eine enthält alle Turtle-Kommandos (Button „Turtle Kommandos“) und die andere Tabelle enthält alle Ausdrücke, die im Bedingungsteil verwendet werden dürfen (Button „Bedingungsteil“). Um eine vorhandene Produktion zu bearbeiten, kann entweder auf die Produktion im Listenfeld doppelgeklickt werden oder die Produktion einmal angeklickt werden und dann über den Button „Bearbeiten“ bearbeitet werden. Um eine Produktion zu löschen, muss diese ausgewählt werden und auf den Button „Löschen“ geklickt werden. Der Button „Alle Löschen“ löscht alle Produktionen.
Nun soll die folgende Produktion eingegeben werden. Für den Produktionskopf schreiben Sie „F“ und in den Produktionskörper (letztes Eingabefeld) schreiben Sie „F-F+F+FF-F-F+F“. Danach klicken Sie auf „Hinzufügen“.
Das eingegebene L-System können Sie auf Schwächen prüfen, dazu klicken Sie einfach auf „Prüfen“. Dies kann gerade bei größeren L-Systemen hilfreich sein. Eine Schwäche in einer Produktion wäre, wenn die Anzahl der öffnenden und schließenden, eckigen oder geschweiften Klammern ungleich ist oder wenn für ein Symbol keine oder mehrere Produktionen existieren. Eine weitere Schwäche liegt vor, wenn die angegebenen Parameter im Produktionskopf nicht verwendet werden, bzw. wenn einige verwendet werden, die nicht im Produktionskopf stehen.
Speichern Sie nun das L-System, indem Sie auf „Speichern“ klicken. Dem L-System können Sie den Namen „Koch“ geben. Jetzt können Sie auf „Bild erzeugen“ klicken. Sie sehen nun die quadratische Kochinsel (ein Fraktal). Sie können nun die Rekursionstiefe auf 3 Stellen und erneut das Bild erzeugen lassen.

G CD-ROM Inhalt

Verzeichnisstruktur	Inhalt
/	Wurzel
Applikationen	Die Applikationen die entstanden sind
Fassade	Kompilierte Fassade und Dokumente dazu
GUI	Kompilierte GUI und Dokumente dazu
Diplomarbeit	Schriftliche Arbeit als PDF und DOC
Abbildungen	Grafiken aus der Diplomarbeit
Literatur	Freie Literaturen zur Diplomarbeit
Eco-Grammar	Über Eco-Grammar
Evolution	Über GA und GP
L-Systeme	Über L-Systeme
P-Systeme	Über P-Systeme
Sonstiges	Enthält sonstige Literaturquellen
Quelltexte	Quelltexte zu den Applikationen der DA
Fassade	Quelltext von Lprocess und Lparser
GUI	Quelltext von der GUI
Resourcen	Programme zum Betrachten der Texte
Software zur Diplomarbeit	Software zur Diplomarbeit
Delphi Ergänzungen	JVCL, DirectX Kapslung, GraphicEx
L-Systeme	Diverse Software zu L-Systemen
POV-Ray	POV-Ray 3.5 für Windows und QuietPOV
Sonstiges	Enthält Imogene

H Quelltexte

Quelltexte von Lprocess:

Dateiname: Lprocess.c

```
/*****************************************************************************************

       Autor:                Jan Derer
       Datum:                09. 06. 04
       Kontakt:              Darkhabits@aol.com
       Programmname:         Lprocess
       Version:              1.0

       Modulename:           Lprocess.c
       Modulversion:         1.0
       Modulbeschreibung:
       In dieser Datei ist die main() Funktion enthalten für das Programm Lprocess.

*****************************************************************************************/


/*****************************************************************************************
                              INCLUDE-DATEIEN
*****************************************************************************************/

#include <stdio.h>
#include "globals.h"
#include "com_line.h"
#include "cfg_file.h"
#include "lparser.h"
#include "pov.h"
#include "misc.h"


/*****************************************************************************************
                              PROTOTYPEN
*****************************************************************************************/

extern void print_Welcome(void);
extern char checkingComline(int *argc, char *argv[]);
extern char checkingCFGFile(void);
extern char convertLP4toLP5(char *old_filename, char *new_filename);
extern char Lparser(void);
extern char Lv2povid(void);
extern char CorrectingPOVFile(void);
extern void startPOV(void);
extern void endPOV(void);
extern char startQPOV(void);
extern char deleteFiles(void);
extern char checkingEnvVariables(char *envp[]);


/*****************************************************************************************

       Funktionsname: main
       Rückgabewert
              Typ:           int
              Bedeutung:     Gibt einen Fehlerstatus zurück (-1) oder
                                    das kein Fehler aufgetreten ist (0).
       Parameter
              Typ:           int
              Name:          argc
              Bedeutung:     Enthält die Anzahl der Argumente von der Kommandozeile.
              Typ:           char **
              Name:          argv
              Bedeutung:     Zeigerarray auf die Strings der Kommandozeile
              Typ:           char **
              Name:          envp
              Bedeutung:     Zeigerarray auf die Strings der Umgebungsvariablen

       Funktionsbeschreibung:
```

```
       Die main()-Funktion delegiert die einzelnen Aufgaben an entsprechende Funktionen wei-
       ter.

******************************************************************************************/

int main(int argc, char *argv[], char *envp[])
{
       char c = 0;

       print_Welcome();

       puts("Checking commandline ...");
       if((c = checkingComline(&argc, argv)) != 1)
              return c;

       /* Wird das Programm nur zum beenden von POV-Ray oder zum konvertieren einer Datei
          aufgerufen, dann endet hier die Abarbeitung schon.*/
       if((info.state & KILL_POV_AND_CONV_ONLY) == KILL_POV_AND_CONV_ONLY)
       {
              puts("Converting Lparser 4 file into Lparser 5 file ...");
              if(convertLP4toLP5(info.filename, info.conv_filename) < 0)
                     return -1;
              endPOV();

              printf("\nFinishing successfully!\n\n");
              return 0;
       }
       else if((info.state & KILL_POV_ONLY) == KILL_POV_ONLY)
       {
              endPOV();
              printf("\nFinishing successfully!\n\n");
              return 0;
       }
       else if((info.state & CONVERT_ONLY) == CONVERT_ONLY)
       {
              puts("Converting Lparser 4 file into Lparser 5 file ...");
              if(convertLP4toLP5(info.filename, info.conv_filename) == -1)
                     return -1;

              printf("\nFinishing successfully!\n\n");
              return 0;
       }

       puts("Checking CFG file ... ");
       if(checkingCFGFile() < 0)
              return -1;

       puts("Checking Environmentvariables ...");
       if(checkingEnvVariables(envp) < 0)
              return -1;

       if((info.state & CONVERT) == CONVERT)
       {
              puts("Converting Lparser 4 file into Lparser 5 file ...");
              if(convertLP4toLP5(info.filename, info.conv_filename) < 0)
                     return -1;
       }

       startPOV();
       puts("Starting Lparser ...");
       if(Lparser() < 0)
              return -1;
       puts("Finishing Lparser");

       puts("Starting LV2POVID ...");
       if(Lv2povid() < 0)
              return -1;
       puts("Finishing LV2POVID");

       puts("Correcting POV File ...");
       if(CorrectingPOVFile() < 0)
              return -1;
       puts("Finishing Correction");

       puts("Starting Render-Process ...");
       if(startQPOV() < 0)
              return -1;
```

```
        if((info.state & KILL_POV) == KILL_POV)
                endPOV();

        if((info.state & KEEP_FILES) == 0)
        {
                puts("Deleting nonrelevant files ...");
                if(deleteFiles() < 0)
                        return -1;
        }

        printf("\nFinishing successfully!\n\n");
        return 0;
}
```

Dateiname: globals.h

```
/*********************************************************************************************

        Autor:                  Jan Derer
        Datum:                  09. 06. 04
        Kontakt:                Darkhabits@aol.com
        Programmname:           Lprocess
        Version:                1.0

        Modulename:             globals.h
        Modulversion:           1.0
        Modulbeschreibung:
        In diesem Modul werden einige Präprozessor-Variablen und zwei Strukturen definiert,
        die in allen Modulen benötigt werden.

*********************************************************************************************/

#ifndef  ___globals_H___

        #define  ___globals_H___

        /* Bitwerte für die state Variable                                              */
        #define CONVERT                         1               /* Entspricht dem Schalter -C */
                                                                /* Konvertieren einer Datei   */
        #define CONVERT_ONLY                    2               /* Entspricht dem Schalter -c */
                                        /* Konvertieren einer Datei und Programm beenden    */
        #define KILL_POV                        4               /* Entspricht dem Schalter -E */
                                                                /* POV-Ray am Ende beenden    */
        #define KILL_POV_ONLY                   8               /* Entspricht dem Schalter -e */
                                        /* Nur POV-Ray beenden und Programm verlassen       */
        #define ANTI_ALIASING                   16              /* Entspricht dem Schalter -A */
                                                /* Anti-Aliasing zur Bilderzeugung nutzen   */
        #define TARGA_FILE                      32              /* Entspricht dem Schalter -T */
                                                        /* Bild als Targa Datei speichern    */
        #define KEEP_FILES                      64              /* Entspricht dem Schalter -F */
                                        /* Dateien aus den Zwischenschritten nicht löschen  */

        #define KILL_POV_AND_CONV_ONLY          10      /* Entspricht dem Schalter -c und -e */
                                        /* Konvertieren einer Datei und POV-Ray beenden     */

        /* Bitwerte für nichterlaubte Schalterkombinationen                             */
        #define ILLEGAL_OPTS_cC                 3               /* Schalterkombination -c -C  */
        #define ILLEGAL_OPTS_cE                 6               /* Schalterkombination -c -E  */
        #define ILLEGAL_OPTS_eC                 9               /* Schalterkombination -e -C  */
        #define ILLEGAL_OPTS_eE                 12              /* Schalterkombination -e -E  */


        /* Alle Anweisungen werden für char Arrays benutzt.                             */
        /* Daher muss zum eigentlich Wert noch +1 gerechnet werden für \0 .             */
        /* Maximale länge des Wertes von Width und Heigt von der Kommandozeile.*/
        #define MAX_VALUE_WIDTH_HEIGHT          11
        #define MAX_FILENAME_LENGTH             51      /* Maximale länge eine Dateinamens.  */
        /* Maximale länge einer Zeile in der CFG Datei.                                 */
        #define MAX_CFG_LINE_LENGTH             151
        /* Maximale länge einer Zeile in einer LS Datei.                            */
        #define MAX_LS_LINE_LENGTH              1001
        /* Maximale Anzahl von Zeichen von der Kommandozeile.                       */
        #define MAX_COMLINE                     128
        /* Maximale länge einer Zeile in einer POV Datei.                           */
        #define MAX_POV_LINE_LENGTH             151
        /* Maximale länge einer Option auf der linken Seite des =, in der CFG Datei.   */
        #define MAX_CFG_LEFTSIDE                11
```

```
        /* Maximale länge einer Option auf der rechten Seite des =, in der CFG Datei     */
        #define MAX_CFG_RIGHTSIDE           141
        /* Maximale Anzahl von Optionen die in einer CFG Datei stehen können.            */
        #define MAX_OPTS_IN_CFG             13

        #define POV_APP              "Pov35MainWinClass"   /* Name der POV-Ray Instanze. */


        /* Die Struktur enthält alle wichtigen Information, von der Kommandozeile und aus der*/
        /* CFG Datei, die alle Teile von Lprocess benötigen, daher gibt es für die Struktur */
        /* eine globale Variable namens info.                                               */
        struct  InfoStruct
        {
                char           filename[MAX_FILENAME_LENGTH],       /* Name einer LS Datei */
                               conv_filename[MAX_FILENAME_LENGTH],  /* Name einer weiteren */
                               /* LS Datei, falls eine konvertierte Datei in einer anderen */
                               /* gespeichert werden soll.                                 */
                               pathPOV[MAX_CFG_RIGHTSIDE],          /* Pfad für POV-RAY   */
                               pathQPOV[MAX_CFG_RIGHTSIDE];         /* Pfad für QuietPOV  */
                unsigned char  state;                       /* Bitweise Speicherung von   */
                                                            /* Optionen für Lprocess      */
                int            height,                      /* Höhe des Bildes            */
                               width;                       /* Breite des Bildes          */
                double         x,                   /* Faktor mit dem die X-Koordinate   */
                                                    /* der Kamera multipliziert wird.    */
                               y,                   /* Faktor mit dem die Y-Koordinate   */
                                                    /* der Kamera multipliziert wird.    */
                               z,                   /* Faktor mit dem die Z-Koordinate   */
                                                    /* der Kamera multipliziert wird.    */
                               light_x,             /* Faktor für die realtive Position  */
                                                    /* der Lichtquelle zur Kamera        */
                               light_y,             /* Faktor für die realtive Position  */
                                                    /* der Lichtquelle zur Kamera        */
                               light_z;             /* Faktor für die realtive Position  */
                                                    /* der Lichtquelle zur Kamera        */
        }   info;


        /* Diese Struktur wird zum auslesen und interpretieren der CFG-Datei benötigt.   */
        struct  CFGStruct
        {       /* Speicherung der linken Seite einer Option. */
                char    left[MAX_CFG_LEFTSIDE];
                /* Speicherung der rechten Seite einer Option. */
                char    right[MAX_CFG_RIGHTSIDE];
        };

#endif
```

Dateiname: com_line.h

```
/*********************************************************************************************

        Autor:                  Jan Derer
        Datum:                  09. 06. 04
        Kontakt:                Darkhabits@aol.com
        Programmname:           Lprocess
        Version:                1.0

        Modulename:             com_line.h
        Modulversion:           1.0
        Modulbeschreibung:
        Header-Datei zum Modul com_line.c.

*********************************************************************************************/

#ifndef  ___com_line_H___

        #define  ___com_line_H___
        extern void print_Welcome(void);
        extern void print_Comline_Help(void);

#endif
```

Dateiname: com_line.c

```
/*****************************************************************************************

       Autor:                Jan Derer
       Datum:                09. 06. 04
       Kontakt:              Darkhabits@aol.com
       Programmname:         Lprocess
       Version:              1.0

       Modulename:           com_line.c
       Modulversion:         1.0
       Modulbeschreibung:
       Wichtige Ausgaben für die Kommandozeile und Funktionen zum verarbeiten der Argumente
       von der Kommandozeile, sind in dem Modul enthalt.

*****************************************************************************************/


/*****************************************************************************************
                              INCLUDE-DATEIEN
*****************************************************************************************/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include "globals.h"


/*****************************************************************************************

       Funktionsname: print_Welcome

       Funktionsbeschreibung:
       Eine einfache Startnachricht für die Konsole.

*****************************************************************************************/

void print_Welcome(void)
{
       puts("-------------------------------------");
       puts("  Lprocess Version 1.0 by Jan Derer");
       puts("  Contact: darkhabits@aol.com");
       printf("-------------------------------------\n\n");
}


/*****************************************************************************************

       Funktionsname: print_Comline_Help

       Funktionsbeschreibung:
       Zeigt auf der Konsole eine kurze Hilfe für Lprocess an.

*****************************************************************************************/

void print_Comline_Help(void)
{
       printf("Usage: lprocess [options] [ls-file to convert] [ls-file]\n\n");
       puts("Examples:");
       puts("lprocess -H640 -W480 -A bekerpl.ls");
       puts("Make a Bitmap-Image of the L-System bekerpl with 640*480 and Anti-Aliasing");
       puts("lprocess -C bekerpl bekerpl5");
       puts("Convert bekerpl and save the L-System to bekerpl5 and generate a Image");
       puts("lprocess -e");
       printf("Just closing the POV-Ray Instance\n\n");
       printf("-?\t\tThis screen\n");
       printf("-H[num]\t\tImage height in pixel\n");
       printf("-W[num]\t\tImage width in pixel\n");
       printf("-T\t\tUsing Imageformat Targa\n");
       printf("-A\t\tUsing Anti-Aliasing\n");
       printf("-C\t\tConverting ls-file to Version 5 and continue\n");
       printf("-c\t\tConverting only\n");
       printf("-E\t\tClose POV-Ray at the end of this process\n");
       printf("-e\t\tClose POV-Ray now\n");
       printf("-F\t\tKeep all intermediate Files\n\n");
```

```
        printf("WARNING: By converting a file without a destination file,\n");
        printf("         the content of the old file will be overwritten!\n\n");
}


/*********************************************************************************************

        Funktionsname: print_Comline_Error
        Parameter
                Typ:            char *
                Name:           errorString
                Bedeutung:      String der die Fehlermeldung enthält.

        Funktionsbeschreibung:
        Sollte bei der Verarbeitung der Argumente der Kommandozeile ein Fehler auftreten, dann
        gibt die Funktion eine formatierte Fehlermeldung aus.

*********************************************************************************************/

void print_Comline_Error(char *errorString)
{
        fprintf(stderr, "ERROR: %s\n", errorString);
        fprintf(stderr, "Usage: lprocess [options] [ls-file to convert] [ls-file]\n");
        fprintf(stderr, "Type -? to get help!\n\n");
}


/*********************************************************************************************

        Funktionsname: checkingState
        Rückgabewert
                Typ:            char
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).

        Funktionsbeschreibung:
        Prüft ob die Argumente der Kommandozeile so gewählt wurden, dass ein reibungsloser Ab-
        lauf garantiert wird.

*********************************************************************************************/

char checkingState(void)
{
        if(((info.state & ILLEGAL_OPTS_cC) == ILLEGAL_OPTS_cC) ||
                ((info.state & ILLEGAL_OPTS_eE) == ILLEGAL_OPTS_eE) ||
                ((info.state & ILLEGAL_OPTS_cE) == ILLEGAL_OPTS_cE) ||
                ((info.state & ILLEGAL_OPTS_eC) == ILLEGAL_OPTS_eC))
        {
                print_Comline_Error("Illegal combination of options!");
                return -1;
        }
        else if(((info.state & KILL_POV_ONLY) == 0) && (strlen(info.filename) == 0))
        {
                print_Comline_Error("No filename found!");
                return -1;
        }
        else if(((info.state & KILL_POV_AND_CONV_ONLY) == KILL_POV_AND_CONV_ONLY) &&
                (strlen(info.filename) == 0))
        {
                print_Comline_Error("Require filename to convert!");
                return -1;
        }

        return 0;
}


/*********************************************************************************************

        Funktionsname: extractValueFromString
        Rückgabewert
                Typ:            int
                Bedeutung:      Gibt die Anzahl der verarbeiteten Zeichen zurück.
                                        Wurde keine Zahl gefunden, wird 0 zurück geliefert.
        Parameter
                Typ:            char *
                Name:           string
                Bedeutung:      String aus dem eine Zahl extrahiert werden soll.
```

```
                Typ:            int
                Name:           index
                Bedeutung:      Index für den String, ab dem die Zahl anfangen soll.
                Typ:            int *
                Name:           result
                Bedeutung:      Zeiger auf eine Variable, in dem der Wert gespeichert werden
                                soll.

        Funktionsbeschreibung:
        Extrahiert aus einem String von der Position index ab, eine Zahl aus dem String, sofern
        eine Zahl vorliegt. Das Ergebnis wird in result abgespeichert. Zurück geliefert wird
        dieAnzahl der verarbeiteten Zeichen.

*********************************************************************************************/

int extractValueFromString(char *string, int index, int *result)
{
        char temp[MAX_VALUE_WIDTH_HEIGHT], c = 0;

        while(isdigit(string[index]))
                temp[c++] = string[index++];

        if(c == 0)
        {
                print_Comline_Error("No Value found!");
                return 0;
        }
        else
        {
                temp[c] = '\0';
                *result = atoi(temp);
        }

        return c;
}


/*********************************************************************************************

        Funktionsname: checkFileExtension
        Parameter
                Typ:            char *
                Name:           filename
                Bedeutung:      String des Dateinamens
                Typ:            char *
                Name:           modfilename
                Bedeutung:      Zeiger auf ein String, in dem der neue Dateiname gespeichert
                                wird.

        Funktionsbeschreibung:
        Diese Funktion überprüft, ob der angegebene Dateiname die Endung .ls enthält. Liegt der
        Fall nicht vor, wird die Endung an den Dateiname rangehangen.

*********************************************************************************************/

void checkFileExtension(char *filename, char *modfilename)
{
        int length = strlen(filename);

        if((filename[length-3] != '.') || (toupper(filename[length-2]) != 'L') ||
                (toupper(filename[length-1]) != 'S'))
                sprintf(modfilename, "%s.ls", filename);
        else
                strcpy(modfilename, filename);
}


/*********************************************************************************************

        Funktionsname: checkingComline
        Rückgabewert
                Typ:            char
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).
        Parameter
                Typ:            int *
                Name:           argc
                Bedeutung:      Enthält die Anzahl der Argumente von der Kommandozeile.
```

```
                Typ:            char **
                Name:           argv
                Bedeutung:      Zeigerarray auf die Strings der Kommandozeile

        Funktionsbeschreibung:
        Die Aufgabe dieser Funktion ist es, die Argumente der Kommandozeile verarbeiten und die
        entsprechenden Variablen mit dem Wert zu setzen.

*******************************************************************************************/

char checkingComline(int *argc, char *argv[])
{
        char    *string                 = NULL, /* Zwischenspeicherung der einzelnen        */
                                                /* Inhalte des Arrays argv.                 */
                temp[MAX_FILENAME_LENGTH], c; /* Zwischenspeicher                         */
        int             i, j,                   /* Zählvariablen                            */
                        k               = 0,    /* Zwischenspeicher                         */
                        length          = 0;    /* Speicherung der Länge eines Strings vom  */
                                                /* Array argv[].                            */

        /* Falls zuwenige Argumente angegeben wurde, wird die Hilfe aufgerufen.          */
        if(*argc == 1)
        {
                print_Comline_Help();
                printf("Need arguments\n\n");
                return 0;
        }

        /* Durchlaufen des Arrays argv[]                                                 */
        for(i = 1; i < *argc; i++)
        {
                string = argv[i];       /* Nächsten String zuweisen zur Verarbeitung.    */

/* Falls der String eine Option enthält, dann wird der String hier weiterverarbeitet.   */
                if(string[0] == '-')
                {

        /* Sollte der String nur das Zeichen enthalten, dann springe zur nächsten Iteration */
                        if((length = strlen(string)) == 1)
                                continue;

                        /* Durchlaufen des Strings und auswerten der einzelnen Zeichen.  */
                        for(j = 1; j < length; j++)
                        {
                                c = string[j];
                                switch(c)
                                {
                                        case 'C':       info.state |= CONVERT;
                                                        break;
                                        case 'c':       info.state |= CONVERT_ONLY;
                                                        break;
                                        case 'E':       info.state |= KILL_POV;
                                                        break;
                                        case 'e':       info.state |= KILL_POV_ONLY;
                                                        break;
                                        case 'A':       info.state |= ANTI_ALIASING;
                                                        break;
                                        case 'T':       info.state |= TARGA_FILE;
                                                        break;
                                        case 'F':       info.state |= KEEP_FILES;
                                                        break;

                /* Falls eine Höhe oder Breite angegeben wird, dann wird aus dem String */
                /* noch die Zahl extrahiert.                                            */
                                        case 'H':       k = extractValueFromString(string, j+1,
                                                                &info.height);
                                                        if(k == 0)
                                                                return -1;
                                                        else
                                                                j += k;
                                                        break;
                                        case 'W':       k = extractValueFromString(string, j+1,
                                                                &info.width);
                                                        if(k == 0)
                                                                return -1;
                                                        else
                                                                j += k;
                                                        break;
```

```
                                case '?':      print_Comline_Help();
                                               return 0;
                                               break;
                                default :print_Comline_Error("Found illegal option!");
                                               return -1;
                                               break;
                                }
                        }
                }

                /* Falls der String keine Optionen enthält, muss es ein Dateiname sein.   */
                else
                {

        /* Wenn mehr als zwei weitere Argumente noch da sind, wird das Programm abgebrochen */
                        if((*argc-i) > 2)
                        {
                                print_Comline_Error("To many filenames or options after
                                                        filename");
                                return -1;
                        }

                /* Liegt nur ein Dateiname vor, dann wird es in info.filename gespeichert. */
                        else if((*argc-i) == 1)
                                strcpy(info.filename, argv[i]);

        /* Andernfalls liegen zwei Dateinamen vor, die entsprechend gespeichert werden. */
                        else
                        {
                                strcpy(info.filename, argv[i]);
                                strcpy(info.conv_filename, argv[++i]);
                                if(info.conv_filename[0] == '-')
                                {
                                print_Comline_Error("To many filenames or options after
                                                        filename");
                                        return -1;
                                }
                                checkFileExtension(info.conv_filename, temp);
                                strcpy(info.conv_filename, temp);
                        }
                        checkFileExtension(info.filename, temp);
                        strcpy(info.filename, temp);
                }
        }

        /* Prüfe die Plausibilität der Eingabe.                                          */
        if(checkingState() == -1)
                return -1;

        return 1;
}
```

Dateiname: cfg_file.h

```
/*********************************************************************************************

        Autor:                  Jan Derer
        Datum:                  09. 06. 04
        Kontakt:                Darkhabits@aol.com
        Programmname:           process
        Version:                1.0

        Modulename:             cfg_file.h
        Modulversion:           1.0
        Modulbeschreibung:
        Header-Datei zum Modul cfg_file.c.

*********************************************************************************************/

#ifndef  ___cfg_file_H___

        #define  ___cfg_file_H___
        extern char checkingCFGFile(void);

#endif
```

Dateiname: cfg_file.c

```
/*****************************************************************************************

       Autor:                Jan Derer
       Datum:                09. 06. 04
       Kontakt:              Darkhabits@aol.com
       Programmname:         process
       Version:              1.0

       Modulename:           cfg_file.c
       Modulversion:         1.0
       Modulbeschreibung:
       Das Modul enthält alle wichtigen Funktion die zum auslesen und interpretieren der
       CFG-Datei benötigt werden.

*****************************************************************************************/


/*****************************************************************************************
                              NCLUDE-DATEIEN
*****************************************************************************************/

#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include "globals.h"


/*****************************************************************************************

       Funktionsname: createDefaultCFGFile

       Funktionsbeschreibung:
       Wurde keine CFG-Datei gefunden, wird eine Datei erzeugt mit Standardwerten und Kommen-
       taren.

*****************************************************************************************/

void createDefaultCFGFile(void)
{
       FILE *CFGFile = NULL;

       CFGFile = fopen("LPROCESS.CFG", "wt");
       fprintf(CFGFile, "// Imageformat: Image=bmp | targa\n");
       fprintf(CFGFile, "Image=bmp\n");
       fprintf(CFGFile, "// Imageheight in pixel\n");
       fprintf(CFGFile, "Height=480\n");
       fprintf(CFGFile, "// Imagewidth in pixel\n");
       fprintf(CFGFile, "Width=640\n");
       fprintf(CFGFile, "// Anti-Aliasing: AA=on | off\n");
       fprintf(CFGFile, "AA=off\n");
       fprintf(CFGFile, "// Path for POV-Ray\n");
       fprintf(CFGFile, "POV=c:\\programme\\pov-ray for windows v3.5\\bin\\\n");
       fprintf(CFGFile, "// Path for QuietPOV\n");
       fprintf(CFGFile, "QPOV=c:\\programme\\pov-ray for windows v3.5\\guiext\\quietpov\\\n");
       fprintf(CFGFile, "// Keep all files? Files=on | off\n");
       fprintf(CFGFile, "Files=off\n");
       fprintf(CFGFile, "// Value to multiplicate with X to get the best cameraposition\n");
       fprintf(CFGFile, "CamX=2\n");
       fprintf(CFGFile, "// Value to multiplicate with Y to get the best cameraposition\n");
       fprintf(CFGFile, "CamY=2\n");
       fprintf(CFGFile, "// Value to multiplicate with Z to get the best cameraposition\n");
       fprintf(CFGFile, "CamZ=2\n");
       fprintf(CFGFile, "// Value to multiplicate with X to get the position for the
                         lightsource\n");
       fprintf(CFGFile, "LightX=1\n");
       fprintf(CFGFile, "// Value to multiplicate with Y to get the position for the
                         lightsource\n");
       fprintf(CFGFile, "LightY=1\n");
       fprintf(CFGFile, "// Value to multiplicate with Z to get the position for the
                         lightsource\n");
       fprintf(CFGFile, "LightZ=1\n");
       fclose(CFGFile);
}
```

```
/*****************************************************************************************

        Funktionsname: getValueFromList
        Rückgabewert
                Typ:            char*
                Bedeutung:      Gibt den String eines bestimmten Wertes aus der Liste zurück.
                                Liegt kein Wert mit der Bezeichnung vor, wird NULL zurück
                                gegeben.
        Parameter
                Typ:            struct CFGStruct []
                Name:           list
                Bedeutung:      Zeiger auf ein Array mit den Wertepaaren aus der CFG-Datei.
                Typ:            char *
                Name:           value
                Bedeutung:      Die Bezeichnung des Wertes nachdem gesucht wird.

        Funktionsbeschreibung:
        Wenn die Wertepaare aus der CFG-Datei eingelesen wurden, kann mit der Funktion über den
        Bezeichner, der Wert als String zurück gegeben werden. Datei wird sequenziell die
        Datenstruktur abgearbeitet.

*****************************************************************************************/

char* getValueFromList(struct CFGStruct list[], char *value)
{
        char i;

        for(i = 0; i < MAX_OPTS_IN_CFG; i++)
                if(stricmp(list[i].left, value) == 0)
                        return list[i].right;

        return NULL;
}


/*****************************************************************************************

        Funktionsname: checkingCFGFile
        Rückgabewert
                Typ:            char
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).

        Funktionsbeschreibung:
        Diese Funktion liest die CFG-Datei ein, füllt die Datenstruktur und interpretiert den
        Inhalt.

*****************************************************************************************/

char checkingCFGFile(void)
{
        FILE            *CFGFile              = NULL; /* Zeiger auf CFG-Datei             */
        char            zeile[MAX_CFG_LINE_LENGTH],   /* Speicher für eine Zeile          */
                        changingSide,                 /* Zeigt an auf welcher Seite       */
                                                      /* man sich gerade befindet.        */
                                                      /* 0 = linke, 1 = rechte Seite      */
                        counter               = 0,    /* Zähler für die Anzahl der        */
                                                      /* gelesene Optionen                */
                        countOK,                      /* Gibt an ob die Zeile eine        */
                                                      /* Option enthält oder nicht.       */
                        *temp                 = NULL, /* Zwischenspeicher für ein String  */
                        path;                         /* Wird gesetzt wenn die Option     */
                                                      /* ein Pfadnamen enthält in dem     */
                                                      /* Leerzeichen mitgespeichert       */
                                                      /* werden.                          */
        int             length                = 0,    /* Speichert die Länge einer        */
                                                      /* gelesene Zeile.                  */
                        i, j,                         /* Zählvariablen                    */
                        errorLine             = 0;    /* Zähler um anzugeben, in welcher  */
                                                      /* Zeile ein Fehler vorliegt.       */
        struct CFGStruct        opts[MAX_OPTS_IN_CFG];/* Datenstruktur zum speichern der  */
                                                      /* Optionen aus der CFG-Datei.      */

/* Existiert eine CFG-Datei? Wenn nicht, dann erstelle eine mit Standardwerten und füllte*/
/* die Variablen auch mit Standardwerten.                                                */
        if((CFGFile = fopen("LPROCESS.CFG", "rt")) == NULL)
        {
                printf("Can't find LPROCESS.CFG! Using default values!\n\n");
```

```
        createDefaultCFGFile();
        info.x = 2.0;
        info.y = 2.0;
        info.z = 2.0;
        if(info.height == 0)
                info.height = 480;
        if(info.width == 0)
                info.width = 640;
        strcpy(info.pathPOV, "\\");
        strcpy(info.pathQPOV, "\\");
        return 0;
}

/* Schleife zum auslesen der CFG-Datei.                                        */
while((!feof(CFGFile)) || (counter == MAX_OPTS_IN_CFG))
{
        errorLine++;
        fgets(zeile, MAX_CFG_LINE_LENGTH, CFGFile);/* Holen einer Zeile aus der Datei*/
        length = strlen(zeile);                    /* Länge des String berechnen.    */
        changingSide = 0;
        j = 0;
        countOK = 0;
        path = 0;

        /* Über die länge des Strings iterieren.                               */
        for(i = 0; i < length; i++)
        {

        /* Wenn die aktuelle und die nächste Position ein / als Zeichen aufweisen, */
        /* dann ist es ein Kommentar. Breche das weitere iterieren der Zeile ab.  */
                if(((i+1) < length) && (zeile[i] == '/') && (zeile[i+1] == '/'))
                {
                        if((changingSide != 0) || (j != 0))
                                countOK = 0;
                        break;
                }

                /* Ist das Zeichen ein Gleichheitszeichen, dann wechsel die Seite.*/
                else if(zeile[i] == '=')
                {
                        changingSide = 1;
                        j = 0;
                }

        /* Speicher Leerzeichen, wenn es eine Pfadoption ist und schon auf die rechte*/
        /* Seite gewechselt wurde.                                             */
                else if(changingSide && path && (zeile[i] == ' '))
                {
                        opts[counter].right[j++] = zeile[i];
                        opts[counter].right[j] = '\0';
                }

        /* Fange die nächste Iteration an, wenn das Zeichen ein Leer-,Tabzeichen oder*/
        /* ein Newline Zeichen ist.                                            */
                else if((zeile[i] == ' ') || (zeile[i] == '\t') || (zeile[i] == '\n'))
                        continue;

                /* Kopiere die Zeichen in die Datenstruktur für die linke Seite.  */
                /* Setze die Pfadvariable, falls der Wert POV oder QPOV ist.      */
                else if((changingSide == 0) && (j < (MAX_CFG_LEFTSIDE-1)))
                {
                        if((j == 0) && ((toupper(zeile[0]) == 'P') ||
                                        (toupper(zeile[0]) == 'Q')))
                        path = 1;
                        countOK = 1;
                        opts[counter].left[j++] = zeile[i];
                        opts[counter].left[j] = '\0';
                }

                /* Kopiere die Zeichen in die Datenstruktur für die rechte Seite. */
                else if(changingSide)
                {
                        opts[counter].right[j++] = zeile[i];
                        opts[counter].right[j] = '\0';
                }
```

```
                /* Falls nichts zu trifft, liegt ein Fehler vor.                */
                else
                {
                        fprintf(stderr, "ERROR: Error in Line %d of LPROCESS.CFG\n\n",
                                errorLine);
                        return -1;
                }
        }

        /* Zähle die Anzahl der Optionen hoch, falls eine Option gefunden wurde.  */
        if(countOK)
                counter++;
}
fclose(CFGFile);

/* Es folgt die Auswertung der CFG-Datei.                                       */
/* Falls per Kommandozeile nicht Anti-Aliasing gesetzt wurde, dann prüfe nach,  */
/* ob in der CFG-Datei die Option gesetzt wurde.                                */
if((info.state & ANTI_ALIASING) == 0)
{
        temp = getValueFromList(opts, "AA");
        if(temp == NULL)
        {
                fprintf(stderr, "ERROR: There exists no value for AA in
                                 LPROCESS.CFG!\n\n");
                return -1;
        }
        else if(stricmp(temp, "on") == 0)
                info.state |= ANTI_ALIASING;
        else if(stricmp(temp, "off") == 0)
                ;
        else
        {
                fprintf(stderr, "ERROR: Illegal Value for AA in LPROCESS.CFG!\n\n");
                return -1;
        }
}

/* Falls per Kommandozeile nicht Targa gesetzt wurde, dann prüfe nach, ob in der */
/* CFG-Datei die Option gesetzt wurde.                                          */
if((info.state & TARGA_FILE) == 0)
{
        temp = getValueFromList(opts, "Image");
        if(temp == NULL)
        {
                fprintf(stderr, "ERROR: There exists no value for Image in
                                 LPROCESS.CFG!\n\n");
                return -1;
        }
        else if(stricmp(temp, "targa") == 0)
                info.state |= TARGA_FILE;
        else if(stricmp(temp, "bmp") == 0)
                ;
        else
        {
                fprintf(stderr, "ERROR: Illegal Value for Image in LPROCESS.CFG!\n\n");
                return -1;
        }
}

/* Wurde über die Kommandozeile keine Höhe gesetzt, dann wird in der            */
/* CFG-Datei nachgeschaut ob ein Wert vorliegt.                                 */
if(info.height == 0)
{
        temp = getValueFromList(opts, "Height");
        if(temp == NULL)
        {
                fprintf(stderr, "ERROR: There exists no value for Height in
                                 LPROCESS.CFG!\n\n");
                return -1;
        }
        else if((info.height = atoi(temp)) < 1)
        {
                fprintf(stderr, "ERROR: Illegal Value for Height in LPROCESS.CFG!\n\n");
                return -1;
        }
}
```

```
	/* Wurde über die Kommandozeile keine Breite gesetzt, dann wird in der		*/
	/* CFG-Datei nachgeschaut ob ein Wert vorliegt.					*/
	if(info.width == 0)
	{
		temp = getValueFromList(opts, "Width");
		if(temp == NULL)
		{
			fprintf(stderr, "ERROR: There exists no value for Width in
					LPROCESS.CFG!\n\n");
			return -1;
		}
		else if((info.width = atoi(temp)) < 1)
		{
			fprintf(stderr, "ERROR: Illegal Value for Width in LPROCESS.CFG!\n\n");
			return -1;
		}
	}

	/* Setze in der info Datenstruktur die Variable x, mit dem Wert aus der CFG-Datei. */
	temp = getValueFromList(opts, "CamX");
	if(temp == NULL)
	{
		fprintf(stderr, "ERROR: There exists no value for CamX in LPROCESS.CFG!\n\n");
		return -1;
	}
	else if((info.x = atof(temp)) == 0.0)
	{
		fprintf(stderr, "ERROR: Illegal Value for CamX in LPROCESS.CFG!\n\n");
		return -1;
	}

	/* Setze in der info Datenstruktur die Variable y, mit dem Wert aus der CFG-Datei. */
	temp = getValueFromList(opts, "CamY");
	if(temp == NULL)
	{
		fprintf(stderr, "ERROR: There exists no value for CamY in LPROCESS.CFG!\n\n");
		return -1;
	}
	else if((info.y = atof(temp)) == 0.0)
	{
		fprintf(stderr, "ERROR: Illegal Value for CamY in LPROCESS.CFG!\n\n");
		return -1;
	}

	/* Setze in der info Datenstruktur die Variable z, mit dem Wert aus der CFG-Datei. */
	temp = getValueFromList(opts, "CamZ");
	if(temp == NULL)
	{
		fprintf(stderr, "ERROR: There exists no value for CamZ in LPROCESS.CFG!\n\n");
		return -1;
	}
	else if((info.z = atof(temp)) == 0.0)
	{
		fprintf(stderr, "ERROR: Illegal Value for CamZ in LPROCESS.CFG!\n\n");
		return -1;
	}

/* Setze in der info Datenstruktur die Variable light_x, mit dem Wert aus der CFG-Datei. */
	temp = getValueFromList(opts, "LightX");
	if(temp == NULL)
	{
		fprintf(stderr, "ERROR: There exists no value for LightX in
				LPROCESS.CFG!\n\n");
		return -1;
	}
	else if((info.light_x = atof(temp)) == 0.0)
	{
		fprintf(stderr, "ERROR: Illegal Value for LightX in LPROCESS.CFG!\n\n");
		return -1;
	}

/* Setze in der info Datenstruktur die Variable light_y, mit dem Wert aus der CFG-Datei. */
	temp = getValueFromList(opts, "LightY");
	if(temp == NULL)
	{
		fprintf(stderr, "ERROR: There exists no value for LightY in
				LPROCESS.CFG!\n\n");
		return -1;
```

```
        }
        else if((info.light_y = atof(temp)) == 0.0)
        {
                fprintf(stderr, "ERROR: Illegal Value for LightY in LPROCESS.CFG!\n\n");
                return -1;
        }

/* Setze in der info Datenstruktur die Variable light_z, mit dem Wert aus der CFG-Datei. */
        temp = getValueFromList(opts, "LightZ");
        if(temp == NULL)
        {
                fprintf(stderr, "ERROR: There exists no value for LightZ in
                                LPROCESS.CFG!\n\n");
                return -1;
        }
        else if((info.light_z = atof(temp)) == 0.0)
        {
                fprintf(stderr, "ERROR: Illegal Value for LightZ in LPROCESS.CFG!\n\n");
                return -1;
        }

        /* Speicher den Pfad, aus der CFG-Datei, für POV-Ray in die Variable pathPOV.    */
        temp = getValueFromList(opts, "POV");
        if(temp == NULL)
        {
                fprintf(stderr, "ERROR: There exists no path for POV-Ray in
                                LPROCESS.CFG!\n\n");
                return -1;
        }
        else
        {
                length = strlen(temp);
                if(length == 1)
                        sprintf(zeile, "pvengine.exe");
                else if(temp[length-1] == '\\')
                        sprintf(zeile, "%spvengine.exe", temp);
                else
                        sprintf(zeile, "%s\\pvengine.exe", temp);
                if((CFGFile = fopen(zeile, "rb")) == NULL)
                {
                        fprintf(stderr, "ERROR: POV-Ray could not be found in this
                                        directory!\n\n");
                        return -1;
                }
                fclose(CFGFile);
                strcpy(info.pathPOV, temp);
        }

        /* Speicher den Pfad, aus der CFG-Datei, für QuietPOV in die Variable pathQPOV.  */
        temp = getValueFromList(opts, "QPOV");
        if(temp == NULL)
        {
                fprintf(stderr, "ERROR: There exists no path for QuietPOV in
                                LPROCESS.CFG!\n\n");
                return -1;
        }
        else
        {
                length = strlen(temp);
                if(length == 1)
                        sprintf(zeile, "quietpov.exe");
                else if(temp[length-1] == '\\')
                        sprintf(zeile, "%squietpov.exe", temp);
                else
                        sprintf(zeile, "%s\\quietpov.exe", temp);
                if((CFGFile = fopen(zeile, "rb")) == NULL)
                {
                        fprintf(stderr, "ERROR: QuietPOV could not be found in this
                                        directory!\n\n");
                        return -1;
                }
                fclose(CFGFile);
                strcpy(info.pathQPOV, temp);
        }
```

```
        /* Falls per Kommandozeile nicht Files gesetzt wurde, dann prüfe nach, ob in der */
        /* CFG-Datei die Option gesetzt wurde.                                            */
        if((info.state & KEEP_FILES) == 0)
        {
                temp = getValueFromList(opts, "Files");
                if(temp == NULL)
                {
                        fprintf(stderr, "ERROR: There exists no value for Files in
                                         LPROCESS.CFG!\n\n");
                        return -1;
                }
                else if(stricmp(temp, "on") == 0)
                        info.state |= KEEP_FILES;
                else if(stricmp(temp, "off") == 0)
                        ;
                else
                {
                        fprintf(stderr, "ERROR: Illegal Value for Files in LPROCESS.CFG!\n\n");
                        return -1;
                }
        }

        return 0;
}
```

Dateiname: misc.h

```
/*********************************************************************************************

        Autor:                  Jan Derer
        Datum:                  09. 06. 04
        Kontakt:                Darkhabits@aol.com
        Programmname:           Lprocess
        Version:                1.0

        Modulename:             misc.h
        Modulversion:           1.0
        Modulbeschreibung:
        Header-Datei zum Modul misc.c.

*********************************************************************************************/

#ifndef  ___misc_H___

        #define  ___misc_H___
        extern char deleteFiles(void);
        extern char checkingEnvVariables(char *envp[]);

#endif
```

Dateiname: misc.c

```
/*********************************************************************************************

        Autor:                  Jan Derer
        Datum:                  09. 06. 04
        Kontakt:                Darkhabits@aol.com
        Programmname:           Lprocess
        Version:                1.0

        Modulename:             misc.c
        Modulversion:           1.0
        Modulbeschreibung:
        Das Modul enthält Funktion die nicht direkt einem anderen Modul zugewiesen werden konn-
        ten.

*********************************************************************************************/


/*********************************************************************************************
                                INCLUDE-DATEIEN
*********************************************************************************************/

#include <stdio.h>
#include <Windows.h>
```

```
#include "globals.h"


/*****************************************************************************************

        Funktionsname: deleteFiles
        Rückgabewert
                Typ:            int
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).

        Funktionsbeschreibung:
        Löscht alle Dateien die in den Zwischenschritten erzeugt wurden.

*****************************************************************************************/

char deleteFiles(void)
{
        char buffer[MAX_COMLINE];

        sprintf(buffer, "DEL *.log");
        if((system(buffer)) > 1)
        {
                fprintf(stderr, "ERROR: Cannot delete Log-Files!\n");
                return -1;
        }

        sprintf(buffer, "DEL *.in?");
        if((system(buffer)) > 1)
        {
                fprintf(stderr, "ERROR: Cannot delete Files!\n");
                return -1;
        }

        sprintf(buffer, "DEL info.txt");
        if((system(buffer)) > 1)
        {
                fprintf(stderr, "ERROR: Cannot delete info.txt!\n");
                return -1;
        }

        sprintf(buffer, "DEL lpar2pov.pov");
        if((system(buffer)) > 1)
        {
                fprintf(stderr, "ERROR: Cannot delete info.txt!\n");
                return -1;
        }

        return 0;
}


/*****************************************************************************************

        Funktionsname: checkingEnvVariables
        Rückgabewert
                Typ:            int
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).
        Parameter
                Typ:            char **
                Name:           envp
                Bedeutung:      Zeigerarray auf die Strings der Umgebungsvariablen

        Funktionsbeschreibung:
        Prüft ob eine Umgebungsvariable für QuietPOV gesetzt wurde und setzt diese gegebenen-
        falls. Zur Prüfung wird der Pfad aus der CFG-Datei mit den Umgebungsvariablen vergli-
        chen.

*****************************************************************************************/

char checkingEnvVariables(char *envp[])
{
        char                    *string;
        int                     i, length,
                                next            = 0,
                                found           = 0;
        unsigned int    j                       = 0;
```

```
	/* Durchsuchen der Umgebungsvariablen nach einem Pfad.                          */
	while((*envp++) != NULL)
	{

		string = *envp;
		if((toupper(string[0]) == 'P') && (toupper(string[1]) == 'A') &&
			(toupper(string[2]) == 'T') && (toupper(string[3]) == 'H') &&
			(toupper(string[4]) == '='))
		{
			length = strlen(string);

			for(i = 5; i < length; i++)
			{
				if(string[i] == info.pathQPOV[j])
				{

					/* Umgebungsvariable wurde gefunden.                    */
					if(j == strlen(info.pathQPOV)-1)
					{
						found = 1;
						break;
					}
					j++;
				}

			/* Überspringe die weitere Überprüfung bis zum nächsten Semikolon. */
				else if(next)
				{
					if(string[i] == ';')
					{
						j = 0;
						next = 0;
					}
				}
			}
			break;
		}
	}

	/* Wenn keine Umgebungsvariable gefunden wurde, dann setze diese jetzt.     */
	if(!found)
	{
		puts("Setting path for QuietPOV ...");
		if(!SetEnvironmentVariable("PATH", info.pathQPOV))
		{
			fprintf(stderr, "ERROR: Cannot setting path for QuietPOV!\n");
			return -1;
		}
	}

	return 0;
}
```

Dateiname: lparser.h

```
/*****************************************************************************************

	Autor:			Jan Derer
	Datum:			09. 06. 04
	Kontakt:		Darkhabits@aol.com
	Programmname:		Lprocess
	Version:		1.0

	Modulename:		lparser.h
	Modulversion:		1.0
	Modulbeschreibung:
	Header-Datei zum Modul lparser.c.

*****************************************************************************************/

#ifndef  ___lparser_H___

	#define  ___lparser_H___
	extern char convertLP4toLP5(char *old_filename, char *new_filename);
	extern char Lparser(void);
	extern char Lv2povid(void);
```

```
        extern char CorrectingPOVFile(void);

#endif
```

Dateiname: lparser.c

```
/*********************************************************************************************

        Autor:                  Jan Derer
        Datum:                  09. 06. 04
        Kontakt:                Darkhabits@aol.com
        Programmname:           Lprocess
        Version:                1.0

        Modulename:             lparser.c
        Modulversion:           1.0
        Modulbeschreibung:
        Das Modul stellt Funktionen bereit zur Konvertierung von LS-Dateien der Version 4 in
        die Version 5, sowie zum Ausführen der Einzelprogramme des Lparser Paketes und zur Be-
        hebung von Fehlern die, die Lparser Programme machen.

*********************************************************************************************/


/*********************************************************************************************
                            INCLUDE-DATEIEN
*********************************************************************************************/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "globals.h"


/*********************************************************************************************

        Funktionsname: convertLP4toLP5
        Rückgabewert
                Typ:            char
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).
        Parameter
                Typ:            char *
                Name:           old_filename
                Bedeutung:      Dateiname der Datei die konvertiert wird.
                Typ:            char *
                Name:           new_filename
                Bedeutung:      Dateiname der Datei die das Ergebnis enthält wird.

        Funktionsbeschreibung:
        Diese Funktion konvertiert eine LS-Datei aus Version 4, in die Version 5.

*********************************************************************************************/

char convertLP4toLP5(char *old_filename, char *new_filename)
{
        FILE    *old_file       = NULL,         /* Zeiger auf die zu konvertierende Datei.   */
                *new_file       = NULL;         /* Zeiger auf die Zieldatei.                 */
        char    same_file       = 0,            /* Wird gesetzt wenn Ziel- und Quelldatei    */
                                                /* die selben sind.                          */
                EndOfFileSymbol= 0,             /* Wird gesetzt wenn das Ende Symbol einer   */
                                                /* Version 4 Datei erreicht wurde.           */
                buffer[MAX_COMLINE],            /* Zwischenspeicher                          */
                old_line[MAX_LS_LINE_LENGTH], /* Speichert die gelesene Zeile.             */
                new_line[MAX_LS_LINE_LENGTH], /* Speichert die konvertierte Zeile.         */
                comment;                        /* Wird gesetzt wenn ein Kommentar anfängt.  */
        int     i, j,                           /* Zählvariablen                             */
                length          = 0;            /* Speichert die Länge einer Zeile.          */

        /* Öffenen der zu konvertierende Datei.                                              */
        if((old_file = fopen(old_filename,"rt")) == NULL)
        {
                fprintf(stderr, "ERROR: File %s cannot be open to convert!\n", old_filename);
                return -1;
        }
```

```
/* Falls in die selbe Datei konvertiert werden soll, dann wird der Inhalt in temp.ls*/
/* zwischengespeichert und später in die Zieldatei umbenannt.                       */
if(new_filename[0] == 0)
{
        same_file = 1;
        new_file = fopen("temp.ls","wt");
}
else
        new_file = fopen(new_filename, "wt");

if(new_file == NULL)
{
        fclose(old_file);
        fprintf(stderr, "ERROR: The new file cannot be open!\n");
        return -1;
}


/* In der Schleife werde die Zeile durchlaufen und entsprechend konvertiert.     */
while((!feof(old_file)) || (EndOfFileSymbol == 0))
{
        fgets(old_line, MAX_LS_LINE_LENGTH, old_file);
        length = strlen(old_line);
        comment = 0;

        for(i = 0, j = 0; i < length; i++, j++)
        {
                switch(old_line[i])
                {
                        case '=':       if(comment) break;    /* Aus = wird ->      */
                                                new_line[j] = '-';
                                                new_line[++j] = '>';
                                                break;

                        case '<':       if(comment) break;    /* Aus < wird \       */
                                                new_line[j] = '\\';
                                                break;

                        case '>':       if(comment) break;    /* Aus > wird /       */
                                                new_line[j] = '/';
                                                break;

                        case '#':       if(comment) break;    /* Aus # wird /*      */
                                                new_line[j] = '/';
                                                new_line[++j] = '*';
                                                break;

                        case '@':       if(i == 0)     /* Endezeichen erreicht      */
                                        {
                                                EndOfFileSymbol = 1;
                                                i = length;
                                                continue;
                                        }

                        default :       new_line[j] = old_line[i];
                }
        }
        if(EndOfFileSymbol == 0)
        {
                new_line[j] = '\0';
                fputs(new_line, new_file);
        }
}

fclose(old_file);
fclose(new_file);

/* Falls Ziel- und Quelldatei die selben sind, dann wird die alte Datei          */
/* gelöscht und temp.ls in die Datei umbenannt.                                  */
if(same_file)
{
        sprintf(buffer, "DEL %s", old_filename);
        if(system(buffer) > 1)
        {
                fprintf(stderr, "ERROR: %s cannot be deleted!\n", old_filename);
                return -1;
        }
        sprintf(buffer, "REN TEMP.LS %s", old_filename);
```

```
                if(system(buffer) > 1)
                {
                        fprintf(stderr, "ERROR: Temp.ls cannot be renamed!\n");
                        return -1;
                }
        }

        return 0;
}


/*********************************************************************************************

        Funktionsname: checkingRulesForBrackets
        Rückgabewert
                Typ:            int
                Bedeutung:      Gibt die Anzahl der gefundenen Fehler zurück.
        Parameter
                Typ:            FILE *
                Name:           LSFile
                Bedeutung:      Zeiger auf die zu durchsuchende Datei.

        Funktionsbeschreibung:
        Diese Funktion prüft ob sich runde Klammern in nicht erlaubten Regelteilen befinden.

*********************************************************************************************/

int checkingRulesForBrackets(FILE *LSFile)
{
        char    line[MAX_LS_LINE_LENGTH];     /* Speicherplatz zum einlesen einer Zeile.  */
        int     i, state, length,     /* state speichert den letzten gefundenen Regelteil,*/
                                      /* dabei steht 1 für linker Kontext, 2 für rechter  */
                                      /* Kontext und 3 für den Bedingungsteil.            */
                errorRule      = 0,   /* Zähler zur Angabe in welcher Regel ein Fehler ist*/
                bracket        = 0,   /* Speichert die Anzahl der gefundenen Klammern.    */
                errorCounter   = 0;   /* Zählt die Anzahl der gefundenen Fehler.          */

        /* Durchlaufe die Schleife bis zum Ende der Datei.                                 */
        while(!feof(LSFile))
        {
                fgets(line, MAX_LS_LINE_LENGTH, LSFile);
                length = strlen(line);
                errorRule++;
                state = 0;

                /* Durchlaufe die Regel zeichenweise.                                      */
                for(i = 0; i < length; i++)
                {

                        /* Wurde eine Klammer gefunden?                                    */
                        if(line[i] == '(' || line[i] == ')')
                                bracket++;

                        /* Linker Kontext gefunden?                                        */
                        else if(state == 0 && line[i] == '<')
                        {

                                /* Wenn Klammern vorhanden, dann gebe eine Fehlermeldung aus! */
                                if(bracket > 0)
                                {
                                        fprintf(stderr, "ERROR: There are brackets in the left
                                                         context of rule %d!\n", errorRule);
                                        errorCounter++;
                                }
                                state++;
                                bracket = 0;
                        }

                        /* Rechter Kontext gefunden?                                       */
                        else if(state < 2 && line[i] == '>')
                        {
                                state = 2;
                                bracket = 0;
                        }

                        /* Bedingungteil gefunden?                                         */
                        else if(line[i] == ':')
                        {
```

```
                /* Wenn Klammern vorhanden (im r. Kontext), dann gebe eine Fehlermeldung aus! */
                        if(state == 2 && bracket > 0)
                        {
                                fprintf(stderr, "ERROR: There are brackets in the right
                                                        context of rule %d!\n", errorRule);
                                errorCounter++;
                        }
                        state = 3;
                        bracket = 0;
                }

                /* Hauptteil der Regel gefunden?                                        */
                else if(line[i] == '-' && line[i+1] == '>')
                {

                        /* Klammern im rechten Kontext gefunden?                        */
                        if(state == 2 && bracket > 0)
                        {
                                fprintf(stderr, "ERROR: There are brackets in the right
                                                        context of rule %d!\n", errorRule);
                                errorCounter++;
                        }

                        /* Klammern im Bedinungsteil gefunden?                          */
                        else if(state == 3 && bracket > 0)
                        {
                                fprintf(stderr, "ERROR: There are brackets in the
                                                      condition of rule %d!\n", errorRule);
                                errorCounter++;
                        }
                        bracket = 0;
                        continue;
                }
        }
    }

    /* Die Anzahl der gefundenen Fehler wird zurück gegeben.                        */
    return errorCounter;
}


/*********************************************************************************************

    Funktionsname: checkingLSFile
    Rückgabewert
            Typ:            int
            Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                das kein Fehler aufgetreten ist (0).

    Funktionsbeschreibung:
    Prüft grob die LS-Datei ob alle wichtigen Werte angegeben wurden, Rekursionstiefe, Axi-
    om, ...

*********************************************************************************************/

char checkingLSFile(void)
{
    FILE    *LSFile         = NULL;             /* Zeiger auf die LS-Datei              */
    char    line[MAX_LS_LINE_LENGTH],     /* Speicherung einer Zeile aus der LS-Datei */
            finishChecking = 0;                 /* Enthält den Status der Prüfung.      */
    int     length, i;

    if(!(LSFile = fopen(info.filename, "rt")))
    {
            fprintf(stderr, "ERROR: The file %s doesn't exists!\n", info.filename);
            return -1;
    }


    /* Die Schleife prüft ob für folgende Parameter entsprechene Werte vorliegene:      */
    /* Rekursionstiefe, Winkel, Anfangsdicke und Axiom                                  */
    while((!feof(LSFile)) || (finishChecking < 4))
    {
            fgets(line, MAX_LS_LINE_LENGTH, LSFile);

            /* Überspringe Kommentarzeilen                                              */
            if(((line[0] == '/') && (line[1] == '*')) || (line[0] == '\n') ||
               (line[0] == '#'))
```

```
                        continue;

                /* Prüfe ob die ersten drei Werte Zahlen und keine Zeichen sind.          */
                else if(finishChecking < 3)
                {
                        length = strlen(line);

                        for(i = 0; i < length; i++)
                        {

                                if((line[i] == ' ')  || (line[i] == '\t') || (line[i] == '\n'))
                                        continue;
                                else if((line[i] > 47) && (line[i] < 58))
                                        break;
                                else
                                {
                                        if(finishChecking == 0)
                                                fprintf(stderr, "ERROR: In %s is not a recursion
                                                        depth define!\n", info.filename);
                                        else if(finishChecking == 1)
                                                fprintf(stderr, "ERROR: In %s is not a basic angel
                                                        define!\n", info.filename);
                                        else
                                                fprintf(stderr, "ERROR: In %s is not a starting
                                                        thickness define!\n", info.filename);
                                        fclose(LSFile);
                                        return -1;
                                }
                        }
                        finishChecking++;
                }

                /* Prüfe ob der vierte Wert ein richtiges Axiom ist und keine Regel.      */
                else if(finishChecking == 3)
                {
                        length = strlen(line);
                        for(i = 0; i < length; i++)
                        {
                                if(((i+1) < length) && (line[i] == '-') && (line[i+1] == '>'))
                                {
                                        fprintf(stderr, "In %s is not a axiom define!\n",
                                                        info.filename);
                                        fclose(LSFile);
                                        return -1;
                                }
                        }
                        finishChecking++;
                }
        }

        /* Prüfen ob Klammern in den Regeln sind.                                         */
        if(checkingRulesForBrackets(LSFile) > 0)
        {
                fclose(LSFile);
                return -1;
        }

        fclose(LSFile);

        return 0;
}


/*********************************************************************************************

        Funktionsname: Lparser
        Rückgabewert
                Typ:            int
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                das kein Fehler aufgetreten ist (0).

        Funktionsbeschreibung:
        Die Aufgabe der Funktion ist es Lparser aufzurufen

*********************************************************************************************/

char Lparser(void)
{
```

```
        FILE    *LOGFile       = NULL;        /* Zeiger auf die LOG-Datei zu Speicherung  */
                                              /* der Meldungen von Lparser.               */
        char    buffer[MAX_COMLINE],          /* Zwischenspeicher                         */
                line[MAX_LS_LINE_LENGTH];     /* Zwischenspeicher für eine Zeile der LOG-Datei*/
        int     linecounter    = 0;           /* Zähler für die gelesenen Zeilen.         */

        strcpy(info.filename, (strlen(info.conv_filename) > 0 ? info.conv_filename :
                                                               info.filename));

        /* Prüfe den Inhalt der LS-Datei.                                                */
        if(checkingLSFile() < 0)
                return -1;

        /* Es wird das vorhandensein der Lparser Datei geprüft.                          */
        if(!(LOGFile = fopen("lparser.exe", "rb")))
        {
                fprintf(stderr, "ERROR: Lparser cannot be found!\n");
                return -1;
        }
        fclose(LOGFile);

        /* Ausführen von Lparser mit den entsprechenden Parametern.                      */
        sprintf(buffer, "Lparser -vc -x%.2f -y%.2f -z%.2f %s > lparser.log", info.x, info.y,
                                                               info.z, info.filename);
        system(buffer);

        /* Lese die LOG-Datei ob ein Fehler aufgetreten ist und melde dies.              */
        if(!(LOGFile = fopen("lparser.log", "rt")))
        {
                fprintf(stderr, "ERROR: Lparser.log doesn't exists!\n");
                return -1;
        }

        while(!feof(LOGFile))
        {
                linecounter++;
                fgets(line, MAX_LS_LINE_LENGTH, LOGFile);
                if((line[0] == 'E') && (line[1] == 'r') && (line[2] == 'r'))
                {
                        fprintf(stderr, "ERROR: There is a error in rule %d !\n",
                                         linecounter-11);
                        return -1;
                }
        }

        fclose(LOGFile);

        return 0;
}


/*********************************************************************************************

        Funktionsname: Lv2povid
        Rückgabewert
                Typ:            int
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).

        Funktionsbeschreibung:
        Lv2povid ruft das gleichnamige Programm auf, um eine POV-Datei zu erzeugen.

*********************************************************************************************/

char Lv2povid(void)
{
        FILE    *file                          = NULL;
        char    buffer[MAX_COMLINE];

        /* Prüfe ob alle wichtigen Dateien vorhanden sind.                               */
        if(!(file = fopen("output.inc", "rt")))
        {
                fprintf(stderr, "ERROR: There is no output.inc file for LV2POVID!\n");
                return -1;
        }
        fclose(file);
```

```
	if(!(file = fopen("info.txt", "rt")))
	{
		fprintf(stderr, "ERROR: There is no info.txt file for LV2POVID!\n");
		return -1;
	}
	fclose(file);

	if(!(file = fopen("lv2povid.cfg", "rt")))
	{
		fprintf(stderr, "ERROR: There is no lv2povid.cfg file for LV2POVID!\n");
		return -1;
	}
	fclose(file);

	/* Rufe LV2POVID auf.                                                        */
	sprintf(buffer, "LV2POVID 2 > lv2povid.log");
	if(system(buffer) > 1)
	{
		fprintf(stderr, "ERROR: Cannot execute LV2POVID!\n");
		return -1;
	}

	return 0;
}

/*****************************************************************************************

	Funktionsname: extractVectorFromLine
	Parameter
		Typ:		char *
		Name:		string
		Bedeutung:	Die Zeile aus der die Daten extrahiert werden sollen.
		Typ:		double *
		Name:		vector
		Bedeutung:	Zeiger auf ein double-Array in dem die Vektordaten gespeichert
				werde.

	Funktionsbeschreibung:
	Extrahiert aus einer Zeile die Informationen für eine Vektor.

*****************************************************************************************/

void extractVectorFromLine(char *string, double *vector)
{
	char	puffer[25],
		VectorPosition		= 0,
		i, j			= 0,
		startRecording		= 0;
	int	len			= strlen(string);

	for(i = 0; i < len; i++)
	{
		if(string[i] == '(')
			startRecording = 1;
		else if(string[i] == ' ' && startRecording == 1)
		{
			puffer[j+1] = '\0';
			vector[VectorPosition++] = atof(puffer);
			j = 0;
		}
		else if(string[i] == ')' && startRecording == 1)
		{
			puffer[j+1] = '\0';
			vector[VectorPosition] = atof(puffer);
			break;
		}
		else if(startRecording == 1)
			puffer[j++] = string[i];
	}
}

/*****************************************************************************************

	Funktionsname: CorrectingPOVFile
	Rückgabewert
		Typ:		char
```

```
               Bedeutung:     Gibt einen Fehlerstatus zurück (-1) oder
                                      das kein Fehler aufgetreten ist (0).

        Funktionsbeschreibung:
        Die von LV2POVID erzeugt POV-Datei enthält fehlerhafte Daten in bezug auf Kameraposition
        und Kameraausrichtung, sowie der Lichtquelleposition. Daher wird aus der INFO.TXT
        die Daten herausgelesen und eine neue POV-Datei erzeugt.

*******************************************************************************************/

char CorrectingPOVFile(void)
{
        FILE    *pov_file                       = NULL;
        FILE    *info_file                      = NULL;
        char    zeile[MAX_POV_LINE_LENGTH];
        short   i;
        double  CamVector[3],
                CenterVector[3];

        if(!(pov_file = fopen("Lpar2pov.pov", "wt")))
        {
                fprintf(stderr, "ERROR: Cannot open Lpar2pov.pov!\n");
                return -1;
        }
        if(!(info_file = fopen("INFO.TXT", "rt")))
        {
                fprintf(stderr, "ERROR: Cannot open info.txt!\n");
                return -1;
        }

        for(i = 0; i < 11; i++)
                fgets(zeile, MAX_POV_LINE_LENGTH, info_file);
        extractVectorFromLine(zeile, CamVector);
        fgets(zeile, MAX_POV_LINE_LENGTH, info_file);
        extractVectorFromLine(zeile, CenterVector);

        fprintf(pov_file, "# include \"colors.inc\"\n");
        fprintf(pov_file, "# include \"textures.inc\"\n");
        fprintf(pov_file, "# include \"shapes.inc\"\n\n");
        fprintf(pov_file, "camera{\n");
        fprintf(pov_file, "  location < %.4f , %.4f , %.4f >\n", CamVector[0], CamVector[2],
                                                                 CamVector[1]);
        fprintf(pov_file, "  look_at < %.4f , %.4f , %.4f >\n", CenterVector[0],
                                                            CenterVector[2], CenterVector[1]);
        fprintf(pov_file, "}\n\n");
        fprintf(pov_file, "#declare col_0  = colour red 1.0 green 1.0 blue 1.0;\n");
        fprintf(pov_file, "#declare col_1  = colour red 0.8 green 0.498039 blue 0.196078;\n");
        fprintf(pov_file, "#declare col_2  = colour red 1.0;\n");
        fprintf(pov_file, "#declare col_3  = colour red 1.0 green 1.0;\n");
        fprintf(pov_file, "#declare col_4  = colour green 1.0;\n");
        fprintf(pov_file, "#declare col_5  = colour blue 1.0 green 1.0;\n");
        fprintf(pov_file, "#declare col_6  = colour blue 1.0;\n");
        fprintf(pov_file, "#declare col_7  = colour red 1.0 blue 1.0;\n");
        fprintf(pov_file, "#declare col_8  = colour red 0.439216 green 0.858824 blue
                                                                    0.576471;\n");
        fprintf(pov_file, "#declare col_9  = colour red 1.0 green 0.498039 blue 0.0;\n");
        fprintf(pov_file, "#declare col_10 = colour red 0.258824 green 0.258824 blue
                                                                    0.435294;\n");
        fprintf(pov_file, "#declare col_11 = colour red 0.6 green 0.196078 blue 0.8;\n");
        fprintf(pov_file, "#declare col_12 = colour red 0.439216 green 0.576471 blue
                                                                    0.858824;\n");
        fprintf(pov_file, "#declare col_13 = colour red 0.556863 green 0.137255 blue
                                                                    0.137255;\n");
        fprintf(pov_file, "#declare col_14 = colour red 0.858824 green 0.858824 blue
                                                                    0.439216;\n");
        fprintf(pov_file, "#declare col_15 = colour red 0.623529 green 0.623529 blue
                                                                    0.372549;\n");
        fprintf(pov_file, "# include \"Lpar2pov.inc\"\n\n");
        fprintf(pov_file, "object { Lsystem }\n\n");
        fprintf(pov_file, "light_source { < %.4f , %.4f , %.4f > color White }\n",
                CamVector[0]*info.light_x, CamVector[2]*info.light_z,
                CamVector[1]*info.light_y);
        //fprintf(pov_file, "light_source { < %.4f , %.4f , %.4f > color Gray60 }\n", );

        fclose(pov_file);
        fclose(info_file);
        return 0;
}
```

Dateiname: pov.h

```
/*****************************************************************************************

        Autor:                  Jan Derer
        Datum:                  09. 06. 04
        Kontakt:                Darkhabits@aol.com
        Programmname:           Lprocess
        Version:                1.0

        Modulename:             pov.h
        Modulversion:           1.0
        Modulbeschreibung:
        Header-Datei zum Modul pov.c.

*****************************************************************************************/

#ifndef  ___pov_H___

        #define  ___pov_H___
        extern void startPOV(void);
        extern void endPOV(void);
        extern char startQPOV(void);

#endif
```

Dateiname: pov.c

```
/*****************************************************************************************

        Autor:                  Jan Derer
        Datum:                  09. 06. 04
        Kontakt:                Darkhabits@aol.com
        Programmname:           Lprocess
        Version:                1.0

        Modulename:             pov.c
        Modulversion:           1.0
        Modulbeschreibung:
        Alle Funktionen die für den Prozess der Bilderzeugung verwendet werden, sind hier ent-
        halten.

*****************************************************************************************/


/*****************************************************************************************
                                INCLUDE-DATEIEN
*****************************************************************************************/

#include <stdio.h>
#include <Windows.h>
#include "globals.h"


/*****************************************************************************************

        Funktionsname: startPOV
        Rückgabewert
                Typ:            int
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).

        Funktionsbeschreibung:
        Prüft ob eine Instanz von POV-Ray schon läuft und startet gegebenenfalls eine Instanz.

*****************************************************************************************/

void startPOV(void)
{
        char    buffer[MAX_COMLINE];
        int     len                     = strlen(info.pathPOV);

        if(len == 1)
                sprintf(buffer, "pvengine.exe");
        else if(info.pathPOV[len-1] == '\\')
                sprintf(buffer, "%spvengine.exe", info.pathPOV);
```

```
        else
                sprintf(buffer, "%s\\pvengine.exe", info.pathPOV);

        if(!FindWindow(POV_APP,0))
        {
                puts("Starting POV-Ray ...");
                WinExec(buffer,SW_HIDE);
        }
}


/*****************************************************************************************

        Funktionsname: endPOV

        Funktionsbeschreibung:
        Es wird geprüft ob eine vorhandenen Instanz von POV-Ray läuft und beendet diese ent-
        sprechend.

*****************************************************************************************/

void endPOV(void)
{
        HWND hPrg;

        if((hPrg = FindWindow(POV_APP,0)) != NULL)
        {
                puts("Closing POV-Ray instance ...");
                SendMessage(hPrg,WM_CLOSE,0,0);
        }
}


/*****************************************************************************************

        Funktionsname: getFilenameWithoutExt
        Parameter
                Typ:            char *
                Name:           filename
                Bedeutung:      String des Dateinamens
                Typ:            char *
                Name:           modfilename
                Bedeutung:      Zeiger auf ein String, in dem der neue Dateiname gespeichert
                                wird.

        Funktionsbeschreibung:
        Liefert einen String vom Dateinamen, nur ohne die Endung.

*****************************************************************************************/

void getFilenameWithoutExt(char *filename, char *modfilename)
{
        char    buffer[MAX_FILENAME_LENGTH];
        int             i, len                = strlen(filename)-3;

        for(i = 0; i < len; i++)
                buffer[i] = filename[i];
        buffer[i] = '\0';

        sprintf(modfilename, "%s", buffer);
}


/*****************************************************************************************

        Funktionsname: createINI
        Rückgabewert
                Typ:            int
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).

        Funktionsbeschreibung:
        Erzeugt eine INI-Datei für POV-Ray mit entsprechend gesetzten Optionen.

*****************************************************************************************/

char createINI(void)
{
```

```
        FILE    *INIFile                            = NULL;
        char    imageFilename[MAX_FILENAME_LENGTH];

        if(!(INIFile = fopen("pov.ini", "wt")))
        {
                fprintf(stderr, "ERROR: POV.INI cannot create for QuietPOV!\n");
                return -1;
        }

        fprintf(INIFile, "Width=%d\n", info.width);
        fprintf(INIFile, "Height=%d\n", info.height);
        fprintf(INIFile, "Display=off\n");

        if((info.state & TARGA_FILE) == TARGA_FILE)
                fprintf(INIFile, "Output_File_Type=T\n");
        else
                fprintf(INIFile, "Output_File_Type=S\n");
        if((info.state & ANTI_ALIASING) == ANTI_ALIASING)
                fprintf(INIFile, "Antialias=on\n");
        else
                fprintf(INIFile, "Antialias=off\n");

        fprintf(INIFile, "Input_File_Name=Lpar2pov.pov\n");
        getFilenameWithoutExt(info.filename, imageFilename);
        fprintf(INIFile, "Output_File_Name=%s\n", imageFilename);

        fclose(INIFile);

        return 0;
}


/*****************************************************************************************

        Funktionsname: startQPOV
        Rückgabewert
                Typ:            int
                Bedeutung:      Gibt einen Fehlerstatus zurück (-1) oder
                                        das kein Fehler aufgetreten ist (0).

        Funktionsbeschreibung:
        Startet QuietPOV, welches wiederum über die GUIExtesion von POV-Ray ein Kommando sen-
        det, zur Erzeugung eines Bildes mit den Optionen, die in der INI-Datei enthalten sind.

*****************************************************************************************/

char startQPOV(void)
{
        FILE    *QPOVFile               = NULL;
        char    buffer[MAX_COMLINE];
        int             len             = strlen(info.pathQPOV);

        if(len == 1)
                sprintf(buffer, "quietpov.exe");
        else if(info.pathQPOV[len-1] == '\\')
                sprintf(buffer, "%squietpov.exe", info.pathQPOV);
        else
                sprintf(buffer, "%s\\quietpov.exe", info.pathQPOV);

        if((QPOVFile = fopen(buffer, "rb")) == NULL)
        {
                fprintf(stderr, "ERROR: QuietPOV cannot be found!\n");
                return -1;
        }
        else
                fclose(QPOVFile);

        puts("Creating INI-File for QuietPOV ...");
        if(createINI() == -1)
                return -1;

        puts("Rendering image ...");
        sprintf(buffer, "quietpov -start pov.ini > pov.log");
        if((system(buffer)) > 1)
        {
                fprintf(stderr, "ERROR: Cannot find QuietPOV!\n");
                return -1;
        }
```

```
        puts("Finishing Rendering ...");

        return 0;
}
```

Quelltext von Lparser (lparser.c):

Die modifizierten Stellen sind fett hervorgehoben.

```
/* -----------------------------------------------------------------------
   LPARSER, L-System Parser/Mutator
   -----------------------------------------------------------------------
   Laurens Lapre
   ljlapre@xs4all.nl
   http://www.xs4all.nl/~ljlapre/

   Modified by Ken Kopp 7/30/2000
   kk82793@ltu.edu

   Modified by Jan Derer 06/09/2004
   darkhabits@aol.com
   -----------------------------------------------------------------------
*/


// --------------------------------- Includes --------------------------------------


#include <stdio.h>
#include <iostream.h>
#include <float.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <math.h>
#include <malloc.h>
#include <ctype.h>
#include <time.h>


////////////////////////////////////////////////////////////////////////////////////////
////
/*            Adding global variables and prototyps of functions                      */
/*                                  by Jan Derer                                      */

double  maxX = 0.0,                 // Saving the maximum positiv value of a x-coordinate
               maxY = 0.0,          // Saving the maximum positiv value of a y-coordinate
               maxZ = 0.0,          // Saving the maximum positiv value of a z-coordinate
               minX = 0.0,          // Saving the maximum negativ value of a x-coordinate
               minY = 0.0,          // Saving the maximum negativ value of a y-coordinate
               minZ = 0.0,          // Saving the maximum negativ value of a z-coordinate
               x, y, z;             // Values to multiply

void createINFOTXT(void);    // Writing INFO.TXT for LV2POVID with the best cameraposition

////////////////////////////////////////////////////////////////////////////////////////
////

// ------------------------------- Constants added by Ken -------------------------


#define PRED_L       30      // Max length of a rule's predecessor, and previous and next
                             // contexts
#define SUCC_L       500     // Max length of a rule's successor string
#define COND_L       30      // Max length of a rule's condition
#define VAR_L        10      // Max length of a variable in a parametric rule

#define MAX_RULES    1000          // Max number of rules in the file
#define MAX_OBJ_LEN  2*1024*1024   // Max length of object (production) string
#define MAX_LIN_LEN  1000          // Max length of a line in the file
#define MAX_VARS     10            // Max number of variables in a parametric rule

#define MAX_EXP_LEN  100      // Max length of an expression in a parametric rule
#define MAX_NUM_LEN  10       // Max length of a real number
```

```
#define MAX_CONST    10       // Max number of constants
#define MAX_SUCC     10       // Max number of successors

#define REC_DEPTH   3         // Depth of recursion


// ---------------------------- Class definitions added by Ken -------------------------


class Rule_Type {
protected:
  char Pred[PRED_L];                // The main block and replacement string
  char *Succ[MAX_SUCC];             // The array of possible successors
  double Probs[MAX_SUCC];           // The probability distribution array
  char *Prev_Ctx, *Next_Ctx;        // Pointers to previous and next contexts
  char *Cond;                       // Pointer to the condition
  int N_Param;                      // The total number of parameters in the rule
  int N_Succ;                       // The number of possible successors

  int Block_Len, Prev_Len;
public:
       Rule_Type(char *Prd, char *Prv, char *Nxt, char *Cnd, char *Suc, double Prb,int N_Par);
       ~Rule_Type();
  void Add_Succ(char *Suc, double Prb);
  bool Same_Rule(char *Prd, char *Prv, char *Nxt, char *Cnd);
       bool Rule_Applies(long i);
       bool Condition_Met(long i);
       int Do_Replace(long i);
       int Prd_Len() { return strlen(Pred); }

  int Get_Pred(char *Str);
  int Get_Pred2(char *Str);
  bool Insert(char *Str, int N_Args, bool App);
  void Replace(char *Str1, char *Str2);
  void Swap_Dirs();
  void Swap_Sizes();
  void Rand_Expr();
};


// ------------------------------ Global variables added by Ken --------------------------


int N_Rules = 0;

char Object_Str[MAX_OBJ_LEN];
char New_Str[MAX_OBJ_LEN];

long Obj_Len;

Rule_Type *Rules[MAX_RULES];

int N_Const = 0;

char Const_Names[MAX_CONST][VAR_L];
char Const_Vals[MAX_CONST][MAX_NUM_LEN];

char Ign_Chars[MAX_NUM_LEN];

/* Basic types ------------------------------------------------------------------------- */


/* Simple types */
#define u8              unsigned char
#define u16             unsigned short int
#define u32             unsigned long int
#define s8              signed char
#define s16             signed short int
#define s32             signed long int
#define r32             float
#define r64             double

/* My own boolean type */
#define boolean         s16
#ifndef TRUE
#define TRUE            (s16) 1
#endif
```

```
#ifndef FALSE
#define FALSE           (s16) 0
#endif


/* Constants ------------------------------------------------------------------------ */


/* Max char size of filename and large string */
#define max_file        512

/* Max vectors per polygon */
#define vectors_p_poly  15

/* Max polygons per object */
#define max_p_object    400

/* Max size of the [] and {} stacks during drawing */
#define max_stack       1024L

/* Version id for the VOL file format */
#define VersionID       11

/* Vector indices */
#define _x              0
#define _y              1
#define _z              2

/* Some often used consts */
#define zero            (r32) 0.0
#define one             (r32) 1.0
#define half_pi         (r32) 1.570796
#define pi              (r32) 3.141592
#define two_pi          (r32) 6.283185
#define LF              ((char) 10)
#define CR              ((char) 13)
#define min_bar         "------------------------------------------------------------"

/* Bounding space for floats */
#define float_min       (r32) -1e30
#define float_max       (r32)  1e30


/* Array and vector types ----------------------------------------------- */


/* A large string */
typedef char            string_file[max_file];

/* Vector arrays */
typedef r32             vector[3];
typedef vector          vectors_4[4];
typedef vector          vectors_max[vectors_p_poly];
typedef r32             float_array[max_p_object + 1];

/* Polygon arrays */
typedef s16             polygon_type[4];
typedef polygon_type    polygon_array[max_p_object + 1];


/* Intrinsics ----------------------------------------------------------- */


#define Abs_s16(A)      ((s16) abs((A)))
#define Abs_s32(A)      ((s32) abs((A)))
#define Abs_r32(A)      ((r32) fabs((A)))
#define Abs_r64(A)      ((r64) fabs((A)))

#define MIN(A,B)        (((A) < (B)) ? (A) : (B))
#define MAX(A,B)        (((A) > (B)) ? (A) : (B))

#define Vector_copy_max_r32(n,A,B) memcpy(((void *) (B)), ((void *) (A)), n*12L)
#define Vector_copy_r32(A,B)       memcpy(((void *) (B)), ((void *) (A)),   12L)
#define Mem_copy(A,B,C)            memcpy(((void *) (B)), ((void *) (A)), (C))
#define Mem_clear(A,B,C)           memset(((void *) (A)), (C), (B))


/* Vector utils and inlines --------------------------------------------- */
```

```
#define Clip_low(a,b)   a = ((a) < (b)) ? (b) : (a)
#define Clip_high(a,b)  a = ((a) > (b)) ? (b) : (a)
#define Clamp(a,b,c)    a = ((a) < (b)) ? (b) : ((a) > (c)) ? (c) : (a)
#define Wrap(a,b,c)     a = ((a) < (b)) ? (a) + (c) : ((a) > (c)) ? (a) - (c) : (a)
#define Lerp(a,b,c)     ((b) + (((c) - (b)) * (a)))
#define Swap(a,b)       {(a) ^= (b); (b) ^= (a); (a) ^= (b);}

#define Vector_length(A)    ((r32) sqrt(   (r32)(A[_x] * A[_x])\
                                         + (r32)(A[_y] * A[_y])\
                                         + (r32)(A[_z] * A[_z]) ))

#define Vector_equal(A,B)   ((A[_x] == B[_x]) && (A[_y] == B[_y]) && (A[_z] == B[_z]))
#define Scalar_product(A,B) (A[_x] * B[_x] + A[_y] * B[_y] + A[_z] * B[_z])

#define Vector_make(A,a,b,c) {\
  A[_x] = a;\
  A[_y] = b;\
  A[_z] = c;\
}

#define Vector_break(A,a,b,c) {\
  a = A[_x];\
  b = A[_y];\
  c = A[_z];\
}

#define Vector_lerp(a,A,B) {\
  A[_x] = Lerp(a, A[_x], B[_x]);\
  A[_y] = Lerp(a, A[_y], B[_y]);\
  A[_z] = Lerp(a, A[_z], B[_z]);\
}

#define Vector_normalize(A)\
{ r32 Dist = (r32) 1.0 / Vector_length(A);\
\
  A[_x] *= Dist;\
  A[_y] *= Dist;\
  A[_z] *= Dist;\
}

#define Vector_copy(A,B) {\
  B[_x] = A[_x];\
  B[_y] = A[_y];\
  B[_z] = A[_z];\
}

#define Vector_product(A,B,C) {\
  C[_x] = A[_y] * B[_z] - A[_z] * B[_y];\
  C[_y] = A[_z] * B[_x] - A[_x] * B[_z];\
  C[_z] = A[_x] * B[_y] - A[_y] * B[_x];\
}

#define Vector_min(A,B,C) {\
  C[_x] = A[_x] - B[_x];\
  C[_y] = A[_y] - B[_y];\
  C[_z] = A[_z] - B[_z];\
}

#define Vector_plus(A,B,C) {\
  C[_x] = A[_x] + B[_x];\
  C[_y] = A[_y] + B[_y];\
  C[_z] = A[_z] + B[_z];\
}

#define Vector_dec(A,B) {\
  A[_x] -= B[_x];\
  A[_y] -= B[_y];\
  A[_z] -= B[_z];\
}

#define Vector_neg(A) {\
  A[_x] = (-A[_x]);\
  A[_y] = (-A[_y]);\
  A[_z] = (-A[_z]);\
}
```

```
#define Vector_inc(A,B) {\
  A[_x] += B[_x];\
  A[_y] += B[_y];\
  A[_z] += B[_z];\
}

#define Vector_plus_fac(A,B,t,C) {\
  C[_x] = A[_x] + (t) * B[_x];\
  C[_y] = A[_y] + (t) * B[_y];\
  C[_z] = A[_z] + (t) * B[_z];\
}

#define Vector_plus_fac2(A,B,b,C,c,D) {\
  D[_x] = A[_x] + (b) * B[_x] + (c) * C[_x];\
  D[_y] = A[_y] + (b) * B[_y] + (c) * C[_y];\
  D[_z] = A[_z] + (b) * B[_z] + (c) * C[_z];\
}

#define Vector_combine(A,a,B,b,C) {\
  C[_x] = (a) * A[_x] + (b) * B[_x];\
  C[_y] = (a) * A[_y] + (b) * B[_y];\
  C[_z] = (a) * A[_z] + (b) * B[_z];\
}

#define Vector_add(A,d) {\
  A[_x] += d;\
  A[_y] += d;\
  A[_z] += d;\
}

#define Vector_sub(A,d) {\
  A[_x] -= d;\
  A[_y] -= d;\
  A[_z] -= d;\
}

#define Vector_div(A,d) {\
  A[_x] /= d;\
  A[_y] /= d;\
  A[_z] /= d;\
}

#define Vector_mul(A,d) {\
  A[_x] *= d;\
  A[_y] *= d;\
  A[_z] *= d;\
}


/* Vector procs ------------------------------------------------------------ */


static vector           M1, M2, M3;         /* The current movetransform matrix */


r32
Do_angle(r32 x1, r32 y1, r32 x2, r32 y2)
{                                           /* Calculate the angle between
                                             * x-axis and x1,y1 -> x2,y2. It
                                             * can handle all kinds of weird
                                             * exceptions */
    r32                     temp, x, y;

    x = x2 - x1;
    y = y2 - y1;

    if (x == zero) {
        if (y < zero)
            temp = -half_pi;
        else
            temp = half_pi;
    } else {
        temp = atan(y / x);
        if (x < zero) {
            if (y < zero)
                temp = -pi + temp;
            else
                temp = pi + temp;
```

```
        }
    }

    if (Abs_r32(temp) < (r32) 0.0001)
        temp = zero;

    if (temp < zero)
        temp += two_pi;
    else if (temp > two_pi)
        temp -= two_pi;

    return temp;
}


void
Move_transform(vector v)
{                                           /* Transform the vector according
                                             * to the current movetransform
                                             * matrix */
    vector                  t;

    Vector_copy_r32(v, t);
    v[_x] = Scalar_product(M1, t);
    v[_y] = Scalar_product(M2, t);
    v[_z] = Scalar_product(M3, t);
}


void
Set_move_transform(r32 a, vector no)
{                                           /* Set a movetransformation matrix
                                             * based on an angle rotation of
                                             * 'a' around the vector 'no' */
    r32                     n11, n22, n33, nxy, nxz, nyz, sina, cosa;

    cosa = (r32) cos(a);
    sina = (r32) sin(a);

    n11 = no[_x] * no[_x];
    n22 = no[_y] * no[_y];
    n33 = no[_z] * no[_z];

    nxy = no[_x] * no[_y];
    nxz = no[_x] * no[_z];
    nyz = no[_y] * no[_z];

    M1[_x] = n11 + (one - n11) * cosa;
    M1[_y] = nxy * (one - cosa) - no[_z] * sina;
    M1[_z] = nxz * (one - cosa) + no[_y] * sina;

    M2[_x] = nxy * (one - cosa) + no[_z] * sina;
    M2[_y] = n22 + (one - n22) * cosa;
    M2[_z] = nyz * (one - cosa) - no[_x] * sina;

    M3[_x] = nxz * (one - cosa) - no[_y] * sina;
    M3[_y] = nyz * (one - cosa) + no[_x] * sina;
    M3[_z] = n33 + (one - n33) * cosa;
}


/* File and conio procs ------------------------------------------------ */


static boolean          native_mode = TRUE;


void
Set_lowhigh(boolean b)
{                                           /* TRUE native Intel mode Low-High,
                                             * FALSE High-Low */
    native_mode = b;
}


void
User_error(char *s,...)
{                                           /* Displays and error messages and
```

```
                                            * exits the program */
    string_file             buf;
    va_list                 args;

    va_start(args, s);
    vsprintf(buf, s, args);
    va_end(args);

    fprintf(stdout, "\n\nError: %s\n\n", buf);
    fflush(stdout);

    exit(EXIT_FAILURE);
}


void
Message(char *s,...)
{                                           /* Sends a message to the output
                                            * stream */
    string_file             buf;
    va_list                 args;

    va_start(args, s);
    vsprintf(buf, s, args);
    va_end(args);

    fprintf(stdout, "%s", buf);
    fflush(stdout);
    fflush(stdout);
}

void
Fget_bin_r32(FILE * f, r32 *val)
{                                           /* Get a r32 value, check for order */
    s32   temp, ta, tb, tc, td;
    r32  *tempr = ((r32 *) ((void *) &temp));

    ta = (s32) getc(f);
    tb = (s32) getc(f);
    tc = (s32) getc(f);
    td = (s32) getc(f);

    if (native_mode) {
        temp = td << 8;
        temp += tc;
        temp = temp << 8;
        temp += tb;
        temp = temp << 8;
        temp += ta;
    } else {
        temp = ta << 8;
        temp += tb;
        temp = temp << 8;
        temp += tc;
        temp = temp << 8;
        temp += td;
    }

    *val = *tempr;
}


void
Fget_bin_s16(FILE * f, s16 *val)
{                                           /* Get a s16 value, check for order */
    s32  ta, tb;

    ta = (s32) getc(f);
    tb = (s32) getc(f);

    if (native_mode) {
        *val = (s16) tb << 8;
        *val += (s16) ta;
    } else {
        *val = (s16) ta << 8;
        *val += (s16) tb;
    }
}
```

```
void
Fget_bin_s32(FILE * f, s32 *val)
{                                       /* Get a s32 value, check for order */
    s32  ta, tb, tc, td;

    ta = (s32) getc(f);
    tb = (s32) getc(f);
    tc = (s32) getc(f);
    td = (s32) getc(f);

    if (native_mode) {
        *val = td << 8;
        *val += tc;
        *val = *val << 8;
        *val += tb;
        *val = *val << 8;
        *val += ta;
    } else {
        *val = ta << 8;
        *val += tb;
        *val = *val << 8;
        *val += tc;
        *val = *val << 8;
        *val += td;
    }
}


void
Fget_bin_u16(FILE * f, u16 *val)
{                                       /* Get a u16 value, check for order */
    s32  ta, tb;

    ta = (s32) getc(f);
    tb = (s32) getc(f);

    if (native_mode) {
        *val = (u16) tb << 8;
        *val += (u16) ta;
    } else {
        *val = (u16) ta << 8;
        *val += (u16) tb;
    }

}


void
Fget_bin_s8(FILE * f, s8 *val)
{
    *val = (s8) getc(f);
}


void
Fget_bin_u8(FILE * f, u8 *val)
{
    *val = (u8) getc(f);
}


void
Fget_bin_u32(FILE * f, u32 *val)
{                                       /* Get a u32 value, check for order */
    s32 ta, tb, tc, td;

    ta = (s32) getc(f);
    tb = (s32) getc(f);
    tc = (s32) getc(f);
    td = (s32) getc(f);

    if (native_mode) {
        *val = td << 8;
        *val += tc;
        *val = *val << 8;
        *val += tb;
```

```
        *val = *val << 8;
        *val += ta;
    } else {
        *val = ta << 8;
        *val += tb;
        *val = *val << 8;
        *val += tc;
        *val = *val << 8;
        *val += td;
    }
}


void
Fput_bin_u8(FILE * f, u8 r)
{
    if (fwrite(&r, sizeof(u8), 1, f) != 1)
        User_error("Can't continue writing outputfile");
}


void
Fput_bin_s8(FILE * f, s8 r)
{
    if (fwrite(&r, sizeof(s8), 1, f) != 1)
        User_error("Can't continue writing outputfile");
}


void
Fput_bin_r32(FILE * f, r32 r)
{
    if (fwrite(&r, sizeof(r32), 1, f) != 1)
        User_error("Can't continue writing outputfile");
}


void
Fput_bin_s16(FILE * f, s16 r)
{
    if (fwrite(&r, sizeof(s16), 1, f) != 1)
        User_error("Can't continue writing outputfile");
}


void
Fput_bin_u16(FILE * f, u16 r)
{
    if (fwrite(&r, sizeof(u16), 1, f) != 1)
        User_error("Can't continue writing outputfile");
}


void
Fput_bin_s32(FILE * f, s32 r)
{
    if (fwrite(&r, sizeof(s32), 1, f) != 1)
        User_error("Can't continue writing outputfile");
}


void
Fput_bin_u32(FILE * f, u32 r)
{
    if (fwrite(&r, sizeof(u32), 1, f) != 1)
        User_error("Can't continue writing outputfile");
}


/* File buffer procs ------------------------------------------------------ */


/* Sinces there is a lot of fileio we use large buffers */
#define f_buffer_size  30L * 1024L
static char            *f_buffer[3] = {NULL, NULL, NULL};


void
```

```
Buffer_IO(FILE * f, s16 i)
{                                           /* Attach filebuffer i to open file */
    setvbuf(f, f_buffer[i], _IOFBF, f_buffer_size);
}


void
Init_file_buf(s16 i)
{                                           /* Init filebuffer i */
    if (f_buffer[i] != NULL)
        return;
    f_buffer[i] = (char *) malloc(f_buffer_size);
    if (f_buffer[i] == NULL)
        User_error("Not enough memory to allocate file buffer");
}


/* Feedback percentage counter ---------------------------------------- */


/* These vars are used to calculate the percentages counters for feedback */
static r32              bar_fac2 = zero;
static s16              old_bar2 = 0;
static u32              bar_max2 = 0;


void
Process_start2(u32 max)
{                                           /* Start bar 2 with the maximum
                                             * value it's going to get */
    bar_fac2 = 100.0 / (r32) max;
    bar_max2 = max;
}


void
Process_update2(u32 now)
{                                           /* Update the percentage counter
                                             * when needed */
    s16 bar = (s16) (bar_fac2 * (r32) now);

    if (bar != old_bar2) {
        old_bar2 = bar;
        Message("\r%3d%%\r", bar);
    }
}


void
Process_end2(void)
{                                           /* Close bar */
    Message("\r    \r");
}


/* Comline procs ------------------------------------------------------ */


#define OPTCHAR '-'

static char             empty[] = "";
static char            *optarg = empty;
static int              optind = 1, opterr = 1;
static char             opts[150] = "";     /* option string */
static s16              s_argc = 0;         /* pointers to comline */
static char           **s_argv = NULL;


static int
getopt(int argc, char *argv[], const char *optstring)
{                                           /* Taken from a source lib
                                             * somewhere */
    static char            *in_pointer = empty;
    char                   *find;

    if (!*in_pointer) {
        if ((!argv[optind]) || (optind >= argc) || (argv[optind][0] != OPTCHAR))
            return -1;
```

```
        in_pointer = argv[optind];
        in_pointer++;
        optind++;
        if (*in_pointer == OPTCHAR) {
            return -1;
        }
    };

    if (*in_pointer == '\0') {
        return 0;
    };

    find = strchr(optstring, *in_pointer);
    if (find == NULL) {
        if (opterr)
            User_error("Option -%c not known", *in_pointer);
        in_pointer = empty;
        return '?';
    };

    if (*(find + 1) == ':') {
        if (!*(in_pointer + 1)) {
            if (optind >= argc) {
                if (opterr)
                    User_error("No argument for option -%c", *in_pointer);
                optarg = empty;
            } else {
                if (argv[optind][0] == OPTCHAR) {
                    if (opterr)
                        User_error("No argument for option -%c but found %s instead",
                                    *in_pointer, argv[optind]);
                }
                optarg = argv[optind++];
            }
        } else {
            optarg = ++in_pointer;
        }
        in_pointer = empty;
    } else {
        optarg = empty;
        in_pointer++;
    }

    return *find;
}


void
Get_comline_opt(char *c, boolean * found, char *result)
{                                           /* Check if comline option 'c' has
                                             * been used and return parameter
                                             * if any */
    int                     f, argc;
    char                  **argv;

    argc = s_argc;
    argv = s_argv;
    optind = 1;

    while ((f = getopt(argc, argv, opts)) != -1) {

        if (f == *c) {
            strcpy(result, optarg);
            *found = TRUE;
            return;
        }
    };

    *found = FALSE;
    strcpy(result, "");
}


void
Get_comline_filename(char *c)
{                                           /* Get the filename argument from
                                             * the comline */
    int                     argc;
```

```
    char                **argv;

    argc = s_argc;
    argv = s_argv;
    optind = 1;

    while (getopt(argc, argv, opts) != -1);

    if (optind == argc)
        User_error("Ran out of arguments before finding file name");

    strcpy(c, argv[optind]);
}


void
Get_comline_progname(char *c)
{                                       /* Get the program name from the
                                         * comline */
    strcpy(c, s_argv[0]);
}


/* Main lparser vars ---------------------------------------------------- */


/* Settings stack used for solving [] references */
typedef struct s_rec {
    vector                  pos;            /* position in 3space of turtle
                                             * origin */
    vector                  fow;            /* forward direction */

    vector                  lef;            /* left direction */
    vector                  upp;            /* up direction */
    vector                  last;           /* last position used for
                                             * connecting cylinders */
    vector                  last_v[9];      /* last vertices of object used for
                                             * connecting cylinders */
    r32                     dis;            /* value of F distance */
    r32                     ang;            /* value of basic angle */
    r32                     thick;          /* value of thickness */
    r32                     dis2;           /* value of Z distance */
    r32                     tr;             /* trope value */
    s16                     col;            /* current color */
    s16                     last_col;       /* color of last object */
} s_rec;

/* Polygon stack used for solving {} references */
typedef struct p_rec {
    s16                     count;          /* number of vertices */
    vector                 *ver;            /* vertex store */
} p_rec;

/* Flags */
static boolean          trope_set = FALSE;  /* see at comline scannign */
static boolean          rand_set = FALSE;
static boolean          user_form = FALSE;
static boolean          closed_form = FALSE;
static boolean          pov_form = FALSE;
static boolean          pov_form2 = FALSE;
static boolean          pov_form3 = FALSE;
static boolean          blb_form = FALSE;
static boolean          inc_out = FALSE;
static boolean          dxf1 = FALSE;
static boolean          dxf2 = FALSE;
static boolean          dxf3 = FALSE;
static boolean          vrml = FALSE;
static boolean          growing = FALSE;    /* real is used for recursion level */
static boolean          last_recur = FALSE; /* processing the last recursion
                                             * step */

/* Init vars */
static r32              zmin = 1e30, thick, min_thick = zero, rand_amount = zero;
static r32              trope_amount = zero;
static u32              polcount = 0;
static u32              poly_limit = 500000L, max_string;
static s16              num = 0, col = 2, lev, last_col = 0;
static r32              dis, ang, dis2, tr = 0.2;
```

```
static vector           sky = {0.0, 0.0, 1.0}, trope;
static vector           last = {1.0, 1.0, 1.0}, last_v[9];
static r32              recursion, fraction;
static vector           axis_x, axis_y, axis_z;

/* Stacks [] and {} */
static s_rec           *stack, org, save;
static s16              scount = 0;
static p_rec           *pstack;
static s16              pscount = 0;

/* Current active transform matrix for drawing */
static vector           C1, C2, C3;

/* Var for rnd */
static r32              rm = (r32) 1.0 / (r32) RAND_MAX;

/* Ouput files */
static FILE            *volume_file = NULL; /* the basic open geometry file */
static FILE            *vf[8];              /* the 8 files when writing
                                             * multiple povray blob files */

/* Object stores */
static polygon_array    poly_store;         /* the store where polygons
                                             * accumulate before saved to disc */
static vector           ver[max_p_object];  /* the store where vertices
                                             * accumulate */
/* Storage of a loaded shape for the -X option */
static char             x_name[max_file];   /* filename of VOL file */
static vector           form_c[max_p_object];   /* vertices */
static polygon_array    form_s;             /* polygons */
static s16              form_ver, form_pol; /* vertices and polygon counts */


/* Check for weird polygons ------------------------------------------- */


 /*
  * Sometimes polygons and/or vertices end up containing floating point
  * exception like NAN etc. These routines find these problems. They can create
  * havoc on input parsers which expect the geometry to be flawless. Typical
  * normalization routines blow up on NAN in vectors.
  */


/* IEEE coded floating point exceptions */
static u32              fp_exp1 = 0x7f800000L;
static u32              fp_exp2 = 0xff800000L;
static u32              fp_exp3 = 0xffc00000L;
static u32              fp_exp4 = 0x7fc00000L;


static                  boolean
Bad_vertex(r32 x, r32 y, r32 z)
{                                           /* Does this vertex contain a
                                             * floation point exception ? */
    union {
        r32                     f;
        u32                     i;
    }                       u;

    u.f = x;
    if ((u.i == fp_exp1) ||
            (u.i == fp_exp2) ||
            (u.i == fp_exp3) ||
            (u.i == fp_exp4)) {
        return TRUE;
    };

    u.f = y;
    if ((u.i == fp_exp1) ||
            (u.i == fp_exp2) ||
            (u.i == fp_exp3) ||
            (u.i == fp_exp4)) {
        return TRUE;
    };

    u.f = z;
```

```
    if ((u.i == fp_exp1) ||
            (u.i == fp_exp2) ||
            (u.i == fp_exp3) ||
            (u.i == fp_exp4)) {
        return TRUE;
    };

    return FALSE;
}


static                  boolean
Invalid_polygon(s16 p)
{                                           /* Can a normal be created on this
                                             * polygon ? */
    vector                  X, Y, Z, N;
    s16                     i;
    r32                     D, x, y, z;

    for (i = 0; i < 4; i++) {
        x = ver[poly_store[p][i]][_x];
        y = ver[poly_store[p][i]][_y];
        z = ver[poly_store[p][i]][_z];
        if (Bad_vertex(x, y, z))
            return TRUE;
    }

    for (i = 0; i < 3; i++) {
        X[i] = ver[poly_store[p][i]][_x];
        Y[i] = ver[poly_store[p][i]][_y];
        Z[i] = ver[poly_store[p][i]][_z];
    }

    N[_x] = Y[_x] * (Z[_y] - Z[_z]) + Y[_y] * (Z[_z] - Z[_x]) + Y[_z] * (Z[_x] - Z[_y]);
    N[_y] = ((-X[_x])) * (Z[_y] - Z[_z]) - X[_y] * (Z[_z] - Z[_x]) - X[_z] * (Z[_x] - Z[_y]);
    N[_z] = X[_x] * (Y[_y] - Y[_z]) + X[_y] * (Y[_z] - Y[_x]) + X[_z] * (Y[_x] - Y[_y]);

    D = N[_x] * N[_x] + N[_y] * N[_y] + N[_z] * N[_z];

    if (D <= (r32) 0.0001)
        return TRUE;
    else
        return FALSE;
}


/* Output data file procs ---------------------------------------------- */


static void
Open_datafile(void)
{                                           /* Open and setup the different
                                             * output files depending on the
                                             * flags */

    char                    S[max_file] = "";
    s16                     i;

    if (pov_form || pov_form2) {
        strcpy(S, inc_out ? "output.inc" : "output.pov");
        Message("Pov file       : %s\n", S);
        volume_file = fopen(S, "wt");
        if (!volume_file)
            User_error("Cannot open file [%s]", S);
        Buffer_IO(volume_file, 0);
               return;

    } else if (pov_form3) {
        for (i = 0; i < 8; i++) {
            sprintf(S, inc_out ? "output%d.inc" : "output%d.pov", i);
            Message("Pov file       : %s\n", S);
            vf[i] = fopen(S, "wt");
            if (!vf[i])
                User_error("Cannot open file [%s]", S);
            fprintf(vf[i], "component 1.0 1.0 <0, 0, 0>\n");
        }
        return;
```

```
} else if (blb_form) {
    strcpy(S, "output.blb");
    Message("Blob file      : %s\n", S);
    volume_file = fopen(S, "wt");
    if (!volume_file)
        User_error("Cannot open file [%s]", S);
    Buffer_IO(volume_file, 0);
    fprintf(volume_file, "[blob]\nThreshold = 0.5\n");
    return;

} else if (dxf1) {
    strcpy(S, "output.dxf");
    Message("Dxf file       : %s\n", S);
    volume_file = fopen(S, "wt");
    if (!volume_file)
        User_error("Cannot open file [%s]", S);
    Buffer_IO(volume_file, 0);

    fprintf(volume_file, "999\nL-System Parser/Mutator\n");
    fprintf(volume_file, "999\nPolyline Polyface Meshes\n");

    if (user_form) {                  /* in this case build a block
                                       * section, include the loaded shape
                                       * and use only block inserts in the
                                       * entities section of the dxf file */

        fprintf(volume_file, "0\nSECTION\n2\nTABLES\n0\nTABLE\n2\nAPPID\n70\n4\n
                              0\nAPPID\n2\nLPARSER\n70\n0\n0\nENDTAB\n0\nENDSEC\n");

        fprintf(volume_file, "0\nSECTION\n2\nBLOCKS\n0\nBLOCK\n8\n0\n2\nBLOCK\n70\n0\n");
        fprintf(volume_file, "10\n0.0\n20\n0.0\n30\n0.0\n3\nBLOCK\n");

        fprintf(volume_file, "0\nPOLYLINE\n66\n1\n8\n0\n62\n0\n70\n64\n");
        fprintf(volume_file, "1001\nLPARSER\n1071\n18500\n1070\n
                              11003\n1000\n<byblock>\n1070\n10999\n");

        for (i = 1; i <= form_ver; i++) {
            fprintf(volume_file, "0\nVERTEX\n8\n0\n62\n0\n");
            fprintf(volume_file, "10\n%g\n", form_c[i][_x]);
            fprintf(volume_file, "20\n%g\n", form_c[i][_y]);
            fprintf(volume_file, "30\n%g\n", form_c[i][_z]);
            fprintf(volume_file, "70\n192\n");
        }

        for (i = 1; i <= form_pol; i++) {
            fprintf(volume_file, "0\nVERTEX\n8\n0\n62\n0\n");
            fprintf(volume_file, "10\n0\n20\n0\n30\n0\n70\n128\n");
            fprintf(volume_file, "71\n%d\n", form_s[i][0]);
            fprintf(volume_file, "72\n%d\n", form_s[i][1]);
            fprintf(volume_file, "73\n%d\n", form_s[i][2]);
            fprintf(volume_file, "74\n%d\n", form_s[i][3]);
        };

        fprintf(volume_file, "0\nSEQEND\n8\n0\n0\nENDBLK\n8\n0\n0\nENDSEC\n");
    };

    fprintf(volume_file, "0\nSECTION\n2\nENTITIES\n");

} else if (dxf2) {
    strcpy(S, "output.dxf");
    Message("Dxf file       : %s\n", S);
    volume_file = fopen(S, "wt");
    if (!volume_file)
        User_error("Cannot open file [%s]", S);
    Buffer_IO(volume_file, 0);

    fprintf(volume_file, "999\nL-System Parser/Mutator\n");
    fprintf(volume_file, "999\n3d Faces List\n");

    fprintf(volume_file, "0\nSECTION\n2\nENTITIES\n");

} else if (dxf3) {
    strcpy(S, "output.raw");
    Message("Raw file       : %s\n", S);
    volume_file = fopen(S, "wt");
    if (!volume_file)
        User_error("Cannot open file [%s]", S);
    Buffer_IO(volume_file, 0);
```

```
    } else if (vrml) {
        strcpy(S, "output.wrl");
        Message("VRML file      : %s\n", S);
        volume_file = fopen(S, "wt");
        if (!volume_file)
            User_error("Cannot open file [%s]", S);
        Buffer_IO(volume_file, 0);

        fprintf(volume_file, "#VRML V1.0 ascii\n");     /* vrml header */
        fprintf(volume_file, "\nSeparator {\n");
        fprintf(volume_file, "\tShapeHints {\n");
        fprintf(volume_file, "\t\tvertexOrdering UNKNOWN_ORDERING\n");
        fprintf(volume_file, "\t\tshapeType UNKNOWN_SHAPE_TYPE\n");
        fprintf(volume_file, "\t\tfaceType CONVEX\n");
        fprintf(volume_file, "\t\tcreaseAngle 0.5\n");
        fprintf(volume_file, "\t}\n");
        fprintf(volume_file, "\tDirectionalLight {\n");
        fprintf(volume_file, "\t\tdirection -0.3 -0.6 -0.9\n");
        fprintf(volume_file, "\t}\n");

    } else {                                /* default output in Lviewer VOL
                                             * format */
        strcpy(S, "output.vol");
        Message("Datafile       : %s\n", S);
        volume_file = fopen(S, "wb");
        if (!volume_file)
            User_error("Cannot open file [%s]", S);
        Buffer_IO(volume_file, 0);

        Fput_bin_u8(volume_file, (u8) VersionID);       /* VOL header */
        Fput_bin_r32(volume_file, (r32) 45.0);
        Fput_bin_r32(volume_file, (r32) 45.0);
        Fput_bin_r32(volume_file, (r32) 90.0);
        Fput_bin_r32(volume_file, (r32) 45.0);
        Fput_bin_r32(volume_file, (r32) 0.0);
        Fput_bin_s16(volume_file, 0);
        Fput_bin_s16(volume_file, 0);
        Fput_bin_s16(volume_file, 100);
        Fput_bin_s16(volume_file, 3000);
        Fput_bin_u16(volume_file, (u16) 0);/* stream format */
    }
}


static void
Close_datafile(void)
{                                           /* Close the different open output
                                             * files */
    s16                     i;

    if (pov_form3) {
        Message("Objects        : %ld\n", polcount);
        for (i = 0; i < 8; i++)
            fclose(vf[i]);
        return;
    };

    if (dxf1 || dxf2)
        fprintf(volume_file, "0\nENDSEC\n0\nEOF\n");

    if (vrml)
        fprintf(volume_file, "}\n");

    Message("Objects        : %ld\n", polcount);
    fclose(volume_file);
}


static void
Save_object(s16 vertices, s16 polygons, s16 color)
{                                           /* Save an object from store to
                                             * disc */
    s32                     t, i, max;

    if (pov_form2 || pov_form3 || blb_form)/* in these case no 'real' geometry
                                            * is saved */
        return;
```

```
    polcount += polygons;

    if (pov_form) {
        for (t = 1; t <= polygons; t++) {

            if (Invalid_polygon(t))
                continue;

            if (poly_store[t][2] == poly_store[t][3]) { /* 3 vertex triangle */
                fprintf(volume_file, "object{triangle{");
                fprintf(volume_file, "<%g, %g, %g>", ver[poly_store[t][0]][_x],
                        ver[poly_store[t][0]][_z], ver[poly_store[t][0]][_y]);
                fprintf(volume_file, "<%g, %g, %g>", ver[poly_store[t][1]][_x],
                        ver[poly_store[t][1]][_z], ver[poly_store[t][1]][_y]);
                fprintf(volume_file, "<%g, %g, %g>}", ver[poly_store[t][2]][_x],
                        ver[poly_store[t][2]][_z], ver[poly_store[t][2]][_y]);

                        fprintf(volume_file, "finish{t_leaf} pigment{color col_%d}}\n",
                                        color % 16);

                        if(maxX < ver[poly_store[t][0]][_x])
                               maxX = ver[poly_store[t][0]][_x];
                        if(maxY < ver[poly_store[t][0]][_y])
                               maxY = ver[poly_store[t][0]][_y];
                        if(maxZ < ver[poly_store[t][0]][_z])
                               maxZ = ver[poly_store[t][0]][_z];
                        if(minX > ver[poly_store[t][0]][_x])
                               minX = ver[poly_store[t][0]][_x];
                        if(minY > ver[poly_store[t][2]][_y])
                               minY = ver[poly_store[t][2]][_y];
                        if(minZ > ver[poly_store[t][0]][_z])
                               minZ = ver[poly_store[t][0]][_z];

                        if(maxX < ver[poly_store[t][1]][_x])
                               maxX = ver[poly_store[t][1]][_x];
                        if(maxY < ver[poly_store[t][1]][_y])
                               maxY = ver[poly_store[t][1]][_y];
                        if(maxZ < ver[poly_store[t][1]][_z])
                               maxZ = ver[poly_store[t][1]][_z];
                        if(minX > ver[poly_store[t][1]][_x])
                               minX = ver[poly_store[t][1]][_x];
                        if(minY > ver[poly_store[t][2]][_y])
                               minY = ver[poly_store[t][2]][_y];
                        if(minZ > ver[poly_store[t][1]][_z])
                               minZ = ver[poly_store[t][1]][_z];

                        if(maxX < ver[poly_store[t][2]][_x])
                               maxX = ver[poly_store[t][2]][_x];
                        if(maxY < ver[poly_store[t][2]][_y])
                               maxY = ver[poly_store[t][2]][_y];
                        if(maxZ < ver[poly_store[t][2]][_z])
                               maxZ = ver[poly_store[t][2]][_z];
                        if(minX > ver[poly_store[t][2]][_x])
                               minX = ver[poly_store[t][2]][_x];
                        if(minY > ver[poly_store[t][2]][_y])
                               minY = ver[poly_store[t][2]][_y];
                        if(minZ > ver[poly_store[t][2]][_z])
                               minZ = ver[poly_store[t][2]][_z];

                  } else {                      /* 4 vertex polygon == 2x triangle */
                fprintf(volume_file, "object{triangle{");
                fprintf(volume_file, "<%g, %g, %g>", ver[poly_store[t][0]][_x],
                        ver[poly_store[t][0]][_z], ver[poly_store[t][0]][_y]);
                fprintf(volume_file, "<%g, %g, %g>", ver[poly_store[t][1]][_x],
                        ver[poly_store[t][1]][_z], ver[poly_store[t][1]][_y]);
                fprintf(volume_file, "<%g, %g, %g>}", ver[poly_store[t][2]][_x],
                        ver[poly_store[t][2]][_z], ver[poly_store[t][2]][_y]);

                        fprintf(volume_file, "finish{t_leaf} pigment{color col_%d}}\n",
color % 16);
                fprintf(volume_file, "object{triangle{");
                fprintf(volume_file, "<%g, %g, %g>", ver[poly_store[t][2]][_x],
                        ver[poly_store[t][2]][_z], ver[poly_store[t][2]][_y]);
                fprintf(volume_file, "<%g, %g, %g>", ver[poly_store[t][3]][_x],
                        ver[poly_store[t][3]][_z], ver[poly_store[t][3]][_y]);
                fprintf(volume_file, "<%g, %g, %g>}", ver[poly_store[t][0]][_x],
                        ver[poly_store[t][0]][_z], ver[poly_store[t][0]][_y]);
```

```
            fprintf(volume_file, "finish{t_leaf} pigment{color col_%d}}\n", color % 16);

                        if(maxX > ver[poly_store[t][0]][_x])
                               maxX = ver[poly_store[t][0]][_x];
                        if(maxY > ver[poly_store[t][0]][_y])
                               maxY = ver[poly_store[t][0]][_y];
                        if(maxZ > ver[poly_store[t][0]][_z])
                               maxZ = ver[poly_store[t][0]][_z];
                        if(minX > ver[poly_store[t][0]][_x])
                               minX = ver[poly_store[t][0]][_x];
                        if(minY > ver[poly_store[t][2]][_y])
                               minY = ver[poly_store[t][2]][_y];
                        if(minZ > ver[poly_store[t][0]][_z])
                               minZ = ver[poly_store[t][0]][_z];

                        if(maxX > ver[poly_store[t][1]][_x])
                               maxX = ver[poly_store[t][1]][_x];
                        if(maxY > ver[poly_store[t][1]][_y])
                               maxY = ver[poly_store[t][1]][_y];
                        if(maxZ > ver[poly_store[t][1]][_z])
                               maxZ = ver[poly_store[t][1]][_z];
                        if(minX > ver[poly_store[t][1]][_x])
                               minX = ver[poly_store[t][1]][_x];
                        if(minY > ver[poly_store[t][2]][_y])
                               minY = ver[poly_store[t][2]][_y];
                        if(minZ > ver[poly_store[t][1]][_z])
                               minZ = ver[poly_store[t][1]][_z];

                        if(maxX > ver[poly_store[t][2]][_x])
                               maxX = ver[poly_store[t][2]][_x];
                        if(maxY > ver[poly_store[t][2]][_y])
                               maxY = ver[poly_store[t][2]][_y];
                        if(maxZ > ver[poly_store[t][2]][_z])
                               maxZ = ver[poly_store[t][2]][_z];
                        if(minX > ver[poly_store[t][2]][_x])
                               minX = ver[poly_store[t][2]][_x];
                        if(minY > ver[poly_store[t][2]][_y])
                               minY = ver[poly_store[t][2]][_y];
                        if(minZ > ver[poly_store[t][2]][_z])
                               minZ = ver[poly_store[t][2]][_z];

                        if(maxX > ver[poly_store[t][3]][_x])
                               maxX = ver[poly_store[t][3]][_x];
                        if(maxY > ver[poly_store[t][3]][_y])
                               maxY = ver[poly_store[t][3]][_y];
                        if(maxZ > ver[poly_store[t][3]][_z])
                               maxZ = ver[poly_store[t][3]][_z];
                        if(minX > ver[poly_store[t][3]][_x])
                               minX = ver[poly_store[t][3]][_x];
                        if(minY > ver[poly_store[t][3]][_y])
                               minY = ver[poly_store[t][3]][_y];
                        if(minZ > ver[poly_store[t][3]][_z])
                               minZ = ver[poly_store[t][3]][_z];

                    }
        }

    } else if (dxf1) {                      /* dxf 3d mesh object */
        fprintf(volume_file, "0\nPOLYLINE\n66\n1\n8\n%d\n70\n64\n", color);

        for (i = 1; i <= vertices; i++) {
            fprintf(volume_file, "0\nVERTEX\n8\n%d\n", color);
            fprintf(volume_file, "10\n%g\n", ver[i][_x]);
            fprintf(volume_file, "20\n%g\n", ver[i][_y]);
            fprintf(volume_file, "30\n%g\n", ver[i][_z]);
            fprintf(volume_file, "70\n192\n");
        }

        for (i = 1; i <= polygons; i++) {
            if (Invalid_polygon(i))
                continue;
            fprintf(volume_file, "0\nVERTEX\n8\n%d\n", color);
            fprintf(volume_file, "10\n0\n20\n0\n30\n0\n70\n128\n");
            fprintf(volume_file, "71\n%d\n", poly_store[i][0]);
            fprintf(volume_file, "72\n%d\n", poly_store[i][1]);
            fprintf(volume_file, "73\n%d\n", poly_store[i][2]);
            fprintf(volume_file, "74\n%d\n", poly_store[i][3]);
```

```
        }

        fprintf(volume_file, "0\nSEQEND\n8\n%d\n", color);

    } else if (dxf2) {                      /* dxf 3dface object */
        for (i = 1; i <= polygons; i++) {

            if (Invalid_polygon(i))
                continue;

            if (poly_store[i][2] == poly_store[i][3]) { /* 3 vertex face */
                fprintf(volume_file, "0\n3DFACE\n8\n%d\n", color);
                fprintf(volume_file, "10\n%g\n", ver[poly_store[i][0]][_x]);
                fprintf(volume_file, "20\n%g\n", ver[poly_store[i][0]][_y]);
                fprintf(volume_file, "30\n%g\n", ver[poly_store[i][0]][_z]);
                fprintf(volume_file, "11\n%g\n", ver[poly_store[i][1]][_x]);
                fprintf(volume_file, "21\n%g\n", ver[poly_store[i][1]][_y]);
                fprintf(volume_file, "31\n%g\n", ver[poly_store[i][1]][_z]);
                fprintf(volume_file, "12\n%g\n", ver[poly_store[i][2]][_x]);
                fprintf(volume_file, "22\n%g\n", ver[poly_store[i][2]][_y]);
                fprintf(volume_file, "32\n%g\n", ver[poly_store[i][2]][_z]);
                fprintf(volume_file, "13\n%g\n", ver[poly_store[i][2]][_x]);
                fprintf(volume_file, "23\n%g\n", ver[poly_store[i][2]][_y]);
                fprintf(volume_file, "33\n%g\n", ver[poly_store[i][2]][_z]);

            } else {                        /* 4 vertex face */
                fprintf(volume_file, "0\n3DFACE\n8\n%d\n", color);
                fprintf(volume_file, "10\n%g\n", ver[poly_store[i][0]][_x]);
                fprintf(volume_file, "20\n%g\n", ver[poly_store[i][0]][_y]);
                fprintf(volume_file, "30\n%g\n", ver[poly_store[i][0]][_z]);
                fprintf(volume_file, "11\n%g\n", ver[poly_store[i][1]][_x]);
                fprintf(volume_file, "21\n%g\n", ver[poly_store[i][1]][_y]);
                fprintf(volume_file, "31\n%g\n", ver[poly_store[i][1]][_z]);
                fprintf(volume_file, "12\n%g\n", ver[poly_store[i][2]][_x]);
                fprintf(volume_file, "22\n%g\n", ver[poly_store[i][2]][_y]);
                fprintf(volume_file, "32\n%g\n", ver[poly_store[i][2]][_z]);
                fprintf(volume_file, "13\n%g\n", ver[poly_store[i][3]][_x]);
                fprintf(volume_file, "23\n%g\n", ver[poly_store[i][3]][_y]);
                fprintf(volume_file, "33\n%g\n", ver[poly_store[i][3]][_z]);
            }
        }

    } else if (vrml) {
        fprintf(volume_file, "\tSeparator {\n");
        fprintf(volume_file, "\t\tMaterial {\n");
        fprintf(volume_file, "\t\t\tdiffuseColor ");

        switch (color) {                    /* translate colors from lparser to
                                             * RGB values */
          case 1:
            fprintf(volume_file, "%f %f %f", 0.3, 0.3, 0.3);
            break;
          case 2:
            fprintf(volume_file, "%f %f %f", 0.8, 0.4, 0.4);
            break;
          case 3:
            fprintf(volume_file, "%f %f %f", 0.8, 0.8, 0.4);
            break;
          case 4:
            fprintf(volume_file, "%f %f %f", 0.4, 0.8, 0.4);
            break;
          case 5:
            fprintf(volume_file, "%f %f %f", 0.4, 0.8, 0.8);
            break;
          case 6:
            fprintf(volume_file, "%f %f %f", 0.4, 0.4, 0.8);
            break;
          case 7:
            fprintf(volume_file, "%f %f %f", 0.8, 0.4, 0.8);
            break;
          case 8:
            fprintf(volume_file, "%f %f %f", 0.2, 0.5, 0.2);
            break;
          case 9:
            fprintf(volume_file, "%f %f %f", 0.2, 0.5, 0.5);
            break;
          case 10:
            fprintf(volume_file, "%f %f %f", 0.2, 0.2, 0.5);
```

```
          break;
        case 11:
          fprintf(volume_file, "%f %f %f", 0.5, 0.2, 0.5);
          break;
        case 12:
          fprintf(volume_file, "%f %f %f", 0.6, 0.2, 0.2);
          break;
        case 13:
          fprintf(volume_file, "%f %f %f", 0.5, 0.5, 0.5);
          break;
        case 14:
          fprintf(volume_file, "%f %f %f", 0.7, 0.7, 0.7);
          break;
        case 15:
          fprintf(volume_file, "%f %f %f", 0.9, 0.9, 0.9);
          break;
        default:
          fprintf(volume_file, "%f %f %f", 0.5, 0.5, 0.5);
          break;
      };

      fprintf(volume_file, "\n\t\t}\n");

  /* Write vertices */
      fprintf(volume_file, "\t\tCoordinate3 {\n");
      fprintf(volume_file, "\t\t\tpoint [\n");
      for (t = 1; t < vertices; t++)
          fprintf(volume_file, "\t\t\t\t%f %f %f,\n", ver[t][_x], ver[t][_z], ver[t][_y]);
      for (t = vertices; t <= vertices; t++)
          fprintf(volume_file, "\t\t\t\t%f %f %f\n", ver[t][_x], ver[t][_z], ver[t][_y]);
      fprintf(volume_file, "\t\t\t]\n");
      fprintf(volume_file, "\t\t}\n");

      fprintf(volume_file, "\t\tMaterialBinding {\n\t\t\tvalue OVERALL\n\t\t}\n");

  /* Write polygons */
      fprintf(volume_file, "\t\tIndexedFaceSet {\n");
      fprintf(volume_file, "\t\t\tcoordIndex [\n");
      for (t = 1; t < polygons; t++) {
          fprintf(volume_file, "\t\t\t\t");
          if (poly_store[t][2] == poly_store[t][3])
              max = 2;
          else
              max = 3;
          for (i = 0; i <= max; i++)
              fprintf(volume_file, "%d, ", poly_store[t][i] - 1);
          fprintf(volume_file, "-1,\n");
      }
      for (t = polygons; t <= polygons; t++) {
          fprintf(volume_file, "\t\t\t\t");
          if (poly_store[t][2] == poly_store[t][3])
              max = 2;
          else
              max = 3;
          for (i = 0; i <= max; i++)
              fprintf(volume_file, "%d, ", poly_store[t][i] - 1);
          fprintf(volume_file, "-1\n");
      }
      fprintf(volume_file, "\t\t\t]\n");
      fprintf(volume_file, "\t\t}\n");

      fprintf(volume_file, "\t}\n");

  } else if (dxf3) {                    /* simple raw format */
      for (i = 1; i <= polygons; i++) {

          if (Invalid_polygon(i))
              continue;

          if (poly_store[i][2] == poly_store[i][3]) { /* 3 vertex triangle */
              fprintf(volume_file, "%g ", ver[poly_store[i][0]][_x]);
              fprintf(volume_file, "%g ", ver[poly_store[i][0]][_y]);
              fprintf(volume_file, "%g ", ver[poly_store[i][0]][_z]);
              fprintf(volume_file, "%g ", ver[poly_store[i][1]][_x]);
              fprintf(volume_file, "%g ", ver[poly_store[i][1]][_y]);
              fprintf(volume_file, "%g ", ver[poly_store[i][1]][_z]);
              fprintf(volume_file, "%g ", ver[poly_store[i][2]][_x]);
              fprintf(volume_file, "%g ", ver[poly_store[i][2]][_y]);
```

```
                fprintf(volume_file, "%g\n", ver[poly_store[i][2]][_z]);

            } else {                        /* 4 vertex polygon = 2x triangle */
                fprintf(volume_file, "%g ", ver[poly_store[i][0]][_x]);
                fprintf(volume_file, "%g ", ver[poly_store[i][0]][_y]);
                fprintf(volume_file, "%g ", ver[poly_store[i][0]][_z]);
                fprintf(volume_file, "%g ", ver[poly_store[i][1]][_x]);
                fprintf(volume_file, "%g ", ver[poly_store[i][1]][_y]);
                fprintf(volume_file, "%g ", ver[poly_store[i][1]][_z]);
                fprintf(volume_file, "%g ", ver[poly_store[i][2]][_x]);
                fprintf(volume_file, "%g ", ver[poly_store[i][2]][_y]);
                fprintf(volume_file, "%g\n", ver[poly_store[i][2]][_z]);
                fprintf(volume_file, "%g ", ver[poly_store[i][0]][_x]);
                fprintf(volume_file, "%g ", ver[poly_store[i][0]][_y]);
                fprintf(volume_file, "%g ", ver[poly_store[i][0]][_z]);
                fprintf(volume_file, "%g ", ver[poly_store[i][2]][_x]);
                fprintf(volume_file, "%g ", ver[poly_store[i][2]][_y]);
                fprintf(volume_file, "%g ", ver[poly_store[i][2]][_z]);
                fprintf(volume_file, "%g ", ver[poly_store[i][3]][_x]);
                fprintf(volume_file, "%g ", ver[poly_store[i][3]][_y]);
                fprintf(volume_file, "%g\n", ver[poly_store[i][3]][_z]);
            }
        }

    } else {                                /* standard VOL output */
        Fput_bin_s16(volume_file, 20);
        Fput_bin_s16(volume_file, vertices);
        Fput_bin_s16(volume_file, polygons);
        Fput_bin_s16(volume_file, color);

        for (t = 1; t <= vertices; t++) {
            Fput_bin_r32(volume_file, ver[t][_x]);
            Fput_bin_r32(volume_file, ver[t][_y]);
            Fput_bin_r32(volume_file, ver[t][_z]);
        }

        for (t = 1; t <= polygons; t++) {
            for (i = 0; i <= 3; i++)
                Fput_bin_s16(volume_file, poly_store[t][i]);
        }
    }
}


/* Add object ------------------------------------------------------------ */


static void
Inverse(vector t, vector v)
{                                           /* Inverse vector transform of a
                                             * matrix built in C123 */
    v[_x] = C1[_x] * t[_x] + C2[_x] * t[_y] + C3[_x] * t[_z];
    v[_y] = C1[_y] * t[_x] + C2[_y] * t[_y] + C3[_y] * t[_z];
    v[_z] = C1[_z] * t[_x] + C2[_z] * t[_y] + C3[_z] * t[_z];
}


static void
Set_ECS(vector n)
{                                           /* Build an ECS transform in the
                                             * axis_xyz vars, used for dxf1
                                             * output */
    vector                  Wy, Wz;
    r32                     fac = (r32) 0.015625;

    Wy[_x] = (r32) 0.0;
    Wy[_y] = (r32) 1.0;
    Wy[_z] = (r32) 0.0;

    Wz[_x] = (r32) 0.0;
    Wz[_y] = (r32) 0.0;
    Wz[_z] = (r32) 1.0;

    Vector_copy_r32(n, axis_z);
    Vector_normalize(axis_z);

    if ((Abs_r32(n[_x]) < fac) && (Abs_r32(n[_y]) < fac)) {
        Vector_product(Wy, axis_z, axis_x);
```

```
    } else {
        Vector_product(Wz, axis_z, axis_x);
    }

    Vector_normalize(axis_x);
    Vector_product(axis_z, axis_x, axis_y);
    Vector_normalize(axis_y);
}


static void
Inverse_ECS(vector p1, vector p2)
{                                           /* Used for dxf1 output */
    p2[_x] = axis_x[_x] * p1[_x] + axis_x[_y] * p1[_y] + axis_x[_z] * p1[_z];
    p2[_y] = axis_y[_x] * p1[_x] + axis_y[_y] * p1[_y] + axis_y[_z] * p1[_z];
    p2[_z] = axis_z[_x] * p1[_x] + axis_z[_y] * p1[_y] + axis_z[_z] * p1[_z];
}


static void
Read_form(void)
{                                           /* Read in and store an external
                                             * object */
    s32                     i, j;
    s16                     ver, pol, k;
    r32                     LR, LE, LT;
    r32                     Dum32;
    s16                     Dum16;

    if (pov_form || pov_form2 || pov_form3 || blb_form)
        return;

    strcat(x_name, ".vol");
    volume_file = fopen(x_name, "rb");
    if (!volume_file)
        User_error("Cannot open file [%s]", x_name);

    Buffer_IO(volume_file, 0);

    Dum16 = getc(volume_file);
    Set_lowhigh((Dum16 == 11));             /* get storage mode */

    Fget_bin_r32(volume_file, &Dum32);      /* header */
    Fget_bin_r32(volume_file, &Dum32);
    Fget_bin_r32(volume_file, &LR);
    Fget_bin_r32(volume_file, &LE);
    Fget_bin_r32(volume_file, &LT);
    fread(&Dum16, sizeof(s16), 1, volume_file);
    fread(&Dum16, sizeof(s16), 1, volume_file);
    fread(&Dum16, sizeof(s16), 1, volume_file);
    fread(&Dum16, sizeof(s16), 1, volume_file);
    fread(&Dum16, sizeof(u16), 1, volume_file);

    form_ver = 0;
    form_pol = 0;
    k = 0;

    for (;;) {
        if (feof(volume_file))
            break;

        Fget_bin_s16(volume_file, &Dum16);
        Fget_bin_s16(volume_file, &ver);    /* vertex count */
        Fget_bin_s16(volume_file, &pol);    /* polygon count */
        Fget_bin_s16(volume_file, &Dum16);

        for (i = 1; i <= ver; i++) {        /* vertices */
            form_ver++;
            Fget_bin_r32(volume_file, &form_c[form_ver][_x]);
            Fget_bin_r32(volume_file, &form_c[form_ver][_y]);
            Fget_bin_r32(volume_file, &form_c[form_ver][_z]);
        }

        for (i = 1; i <= pol; i++) {        /* polygons */
            form_pol++;
            for (j = 0; j <= 3; j++)
                Fget_bin_s16(volume_file, &form_s[form_pol][j]);
            for (j = 0; j <= 3; j++)
```

```
                form_s[form_pol][j] += k;
        }

        k = form_ver;
    }

    fclose(volume_file);
}


static void
Define_form(vector p1, vector p2, vector up, s16 c)
{                                           /* Insert external object. The
                                             * basis is to create the resulting
                                             * normalized direction vectors
                                             * from the turtle vectors and use
                                             * this a matrix for inverse
                                             * transforming the object from its
                                             * location round the origin to its
                                             * location at the turtle origin */
    vector                  dis, d1, d2, d3, in, ext, Q, P;
    s16                     i;
    r32                     s, d, r1, r2;
    char                    layer[10] = "";

 /* p1 = location, p2 = forward, up = up */
    zmin = MIN(zmin, p1[_z]);
    zmin = MIN(zmin, p2[_z]);

 /* setup */
    Vector_min(p2, p1, dis);
    d = Vector_length(dis);
    if (d == (r32) 0.0)
        return;
    s = d * thick;
    s = (s < (r32) min_thick) ? min_thick : s;

 /* d1 */
    Vector_copy_r32(dis, d1);
    Vector_normalize(d1);

    if (dxf1) {                             /* setup ECS and insert the block
                                             * reference */
        Set_ECS(d1);
        Inverse_ECS(p1, d2);
        Vector_copy(d1, ext);
        Vector_copy(d2, in);
        sprintf(layer, "%d", c);
        fprintf(volume_file, "0\nINSERT\n8\n%s\n2\nBLOCK\n", layer);
        fprintf(volume_file, "10\n%g\n20\n%g\n30\n%g\n", in[_x], in[_y], in[_z]);
        fprintf(volume_file, "41\n%g\n42\n%g\n43\n%g\n", s, s, d);
        fprintf(volume_file, "210\n%g\n220\n%g\n230\n%g\n", ext[_x], ext[_y], ext[_z]);
        polcount++;
        return;
    };

 /* d2 */
    Vector_copy_r32(up, d2);
    Vector_normalize(d2);

 /* d3 */
    Vector_product(d1, d2, d3);
    Vector_normalize(d3);

 /* setup transform */
    Vector_copy_r32(d3, C1);                /* new x-axis */
    Vector_copy_r32(d2, C2);                /* new y-axis */
    Vector_copy_r32(d1, C3);                /* new z-axis */

    if (pov_form) {                         /* insert a reference to l_base */
        d *= 0.7;
        s *= 0.7;
        r1 = 57.0 * Do_angle(0.0, 0.0, d1[_z], sqrt(d1[_x] * d1[_x] + d1[_y] * d1[_y]));
        r2 = 57.0 * Do_angle(0.0, 0.0, d1[_x], d1[_y]);
        fprintf(volume_file, "object{l_base ");
        fprintf(volume_file, "finish{t_base} pigment{color col_%d}", c % 16);
        fprintf(volume_file, "scale<%g, %g, %g>", s, d, s);
        fprintf(volume_file, "rotate<%g, %g, %g>", 0.0, 0.0, -r1);
```

```
        fprintf(volume_file, "rotate<%g, %g, %g>", 0.0, -r2, 0.0);
        fprintf(volume_file, "translate<%g, %g, %g>}\n", p1[_x], p1[_z], p1[_y]);

        polcount++;

               if(maxX < p1[_x])
                      maxX = p1[_x];
               if(maxY < p1[_y])
                      maxY = p1[_y];
               if(maxZ < p1[_z])
                      maxZ = p1[_z];
               if(minX > p1[_x])
                      minX = p1[_x];
               if(minY > p1[_y])
                      minY = p1[_y];
               if(minZ > p1[_z])
                      minZ = p1[_z];

        return;

    } else if (pov_form2) {                /* write out a blob origin */
        fprintf(volume_file, "component 1.0 %g <%g, %g, %g>\n", d, p1[_x], p1[_z], p1[_y]);
        polcount++;
        return;

    } else if (pov_form3) {                /* write out a blob origin to the
                                            * file based on the color */
        fprintf(vf[c % 8], "component 1.0 %g <%g, %g, %g>\n", d, p1[_x], p1[_z], p1[_y]);
        polcount++;
        return;

    } else if (blb_form) {                 /* write out a blob in BLB format */
        fprintf(volume_file, "Sphere = %g %g %g 1.0 %g\n", p1[_x], p1[_z], p1[_y], d);
        polcount++;
        return;
    };

 /* Else use the good old Lviewer VOL format */
    for (i = 1; i <= form_ver; i++) {      /* vertices */
        Q[_x] = form_c[i][_x] * s;
        Q[_y] = form_c[i][_y] * s;
        Q[_z] = form_c[i][_z] * d;
        Inverse(Q, P);
        Vector_plus(P, p1, ver[i]);
    }

    for (i = 1; i <= form_pol; i++) {      /* polygons */
        poly_store[i][0] = form_s[i][0];
        poly_store[i][1] = form_s[i][1];
        poly_store[i][2] = form_s[i][2];
        poly_store[i][3] = form_s[i][3];
    }

    Save_object(form_ver, form_pol, c);    /* save the stored object */
}


static void
Define_block(vector p1, vector p2, vector up, s16 c)
{                                           /* Insert basic block. Here we
                                             * build a cube shape directly on
                                             * the input vectors. */
    vector                  dis, d1, d2, d3;
    s16                     i;
    r32                     s, d;

    if (pov_form || pov_form2 || pov_form3 || blb_form)
        return;

    zmin = MIN(zmin, p1[_z]);
    zmin = MIN(zmin, p2[_z]);

 /* setup */
    Vector_min(p2, p1, dis);
    d = Vector_length(dis);
    if (d == (r32) 0.0)
        return;
    s = d * thick;
```

```
    s = (s < (r32) min_thick) ? min_thick : s;
    s *= 0.5;

 /* d1 */
    Vector_copy_r32(dis, d1);
    Vector_normalize(d1);

 /* d2 */
    Vector_copy_r32(up, d2);
    Vector_normalize(d2);

 /* d3 */
    Vector_product(d1, d2, d3);
    Vector_normalize(d3);

 /* base 1, 3 */
    Vector_plus(d2, d3, d1);
    Vector_normalize(d1);
    Vector_plus_fac(p1, d1, s, ver[1]);
    Vector_plus_fac(p1, d1, -s, ver[3]);

 /* base 2, 4 */
    Vector_min(d2, d3, d1);
    Vector_normalize(d1);
    Vector_plus_fac(p1, d1, s, ver[2]);
    Vector_plus_fac(p1, d1, -s, ver[4]);

 /* end */
    for (i = 1; i <= 4; i++)
        Vector_plus(ver[i], dis, ver[i + 4]);

 /* polygons */
    poly_store[1][0] = 1;
    poly_store[1][1] = 5;
    poly_store[1][2] = 6;
    poly_store[1][3] = 2;

    poly_store[2][0] = 2;
    poly_store[2][1] = 6;
    poly_store[2][2] = 7;
    poly_store[2][3] = 3;

    poly_store[3][0] = 3;
    poly_store[3][1] = 7;
    poly_store[3][2] = 8;
    poly_store[3][3] = 4;

    poly_store[4][0] = 4;
    poly_store[4][1] = 8;
    poly_store[4][2] = 5;
    poly_store[4][3] = 1;

    poly_store[5][0] = 1;
    poly_store[5][1] = 2;
    poly_store[5][2] = 3;
    poly_store[5][3] = 4;

    poly_store[6][0] = 8;
    poly_store[6][1] = 7;
    poly_store[6][2] = 6;
    poly_store[6][3] = 5;

    Save_object(8, 6, c);
}


static void
Define_closed(vector p1, vector p2, vector up, s16 c)
{                                           /* Insert connected cylinder shape.
                                             * The lastxxx vars are used to
                                             * store the previous top of the
                                             * cylinder for conntecing a next
                                             * one. Since the vars are stacked
                                             * for [] we can connect correctly
                                             * according to current nesting
                                             * level. */
    vector                  dis, d1, d2, d3, t1, t2;
    s16                     i, ii, col;
```

```
    r32                     s, d, dd = float_max;

    if (pov_form || pov_form2 || pov_form3 || blb_form)
        return;

    zmin = MIN(zmin, p1[_z]);
    zmin = MIN(zmin, p2[_z]);

 /* setup */
    Vector_min(p2, p1, dis);
    d = Vector_length(dis);
    if (d == (r32) 0.0)
        return;
    s = d * thick;
    s = (s < (r32) min_thick) ? min_thick : s;
    s *= 0.5;

 /* d1 */
    Vector_copy_r32(dis, d1);
    Vector_normalize(d1);

 /* d2 */
    Vector_copy_r32(up, d2);
    Vector_normalize(d2);

 /* d3 */
    Vector_product(d1, d2, d3);
    Vector_normalize(d3);

    Vector_plus(d2, d3, t1);
    Vector_normalize(t1);
    Vector_min(d2, d3, t2);
    Vector_normalize(t2);

    Vector_plus_fac(p1, t1, s, ver[1]);
    Vector_plus_fac(p1, t1, -s, ver[5]);
    Vector_plus_fac(p1, t2, s, ver[3]);
    Vector_plus_fac(p1, t2, -s, ver[7]);

    s *= (r32) 0.7071;
    Vector_plus_fac2(p1, t1, s, t2, s, ver[2]);
    Vector_plus_fac2(p1, t1, -s, t2, s, ver[4]);
    Vector_plus_fac2(p1, t1, -s, t2, -s, ver[6]);
    Vector_plus_fac2(p1, t1, s, t2, -s, ver[8]);

 /* end */
    for (i = 1; i <= 8; i++)
        Vector_plus(ver[i], dis, ver[i + 8]);

    if (last_col == c) {
        Vector_min(p1, last, dis);
        d = Vector_length(dis);

        if (d < (r32) 1.0) {
            for (i = 1; i <= 8; i++) {
                Vector_min(ver[1], last_v[i], dis);
                d = Vector_length(dis);
                if (d < dd) {
                    dd = d;
                    ii = i;
                }
            }
            for (i = 1; i <= 8; i++) {
                Vector_copy_r32(last_v[ii], ver[i]);
                ii = (ii + 1) % 9;
                if (ii == 0)
                    ii = 1;
            }
        }
    };

 /* polygons */
    poly_store[1][0] = 1;
    poly_store[1][1] = 9;
    poly_store[1][2] = 10;
    poly_store[1][3] = 2;

    poly_store[2][0] = 2;
```

```
    poly_store[2][1] = 10;
    poly_store[2][2] = 11;
    poly_store[2][3] = 3;

    poly_store[3][0] = 3;
    poly_store[3][1] = 11;
    poly_store[3][2] = 12;
    poly_store[3][3] = 4;

    poly_store[4][0] = 4;
    poly_store[4][1] = 12;
    poly_store[4][2] = 13;
    poly_store[4][3] = 5;

    poly_store[5][0] = 5;
    poly_store[5][1] = 13;
    poly_store[5][2] = 14;
    poly_store[5][3] = 6;

    poly_store[6][0] = 6;
    poly_store[6][1] = 14;
    poly_store[6][2] = 15;
    poly_store[6][3] = 7;

    poly_store[7][0] = 7;
    poly_store[7][1] = 15;
    poly_store[7][2] = 16;
    poly_store[7][3] = 8;

    poly_store[8][0] = 8;
    poly_store[8][1] = 16;
    poly_store[8][2] = 9;
    poly_store[8][3] = 1;

    Save_object(16, 8, c);

    last_col = c;
    Vector_copy_r32(p2, last);
    for (i = 1; i <= 8; i++)
        Vector_copy_r32(ver[i + 8], last_v[i]);
}


static void
Ground_plane(void)
{                                           /* Add a simple large groundplane */
    r32                     l = (r32) 1e5;

    ver[1][_x] = -l;
    ver[1][_y] = l;
    ver[1][_z] = zmin;

    ver[2][_x] = l;
    ver[2][_y] = l;
    ver[2][_z] = zmin;

    ver[3][_x] = l;
    ver[3][_y] = -l;
    ver[3][_z] = zmin;

    ver[4][_x] = -l;
    ver[4][_y] = -l;
    ver[4][_z] = zmin;

    poly_store[1][0] = 4;
    poly_store[1][1] = 3;
    poly_store[1][2] = 2;
    poly_store[1][3] = 1;

    Save_object(4, 1, 1);
}


/* L-system routines ------------------------------------------------- */


static r32
Rnd(void)
```

```
{                                               /* Get a random number */
    return (r32) rand() * rm;
}



#define Util_t(In,C1,C2,C3,Out) {\
  Out[_x] = Scalar_product(C1,In);\
  Out[_y] = Scalar_product(C2,In);\
  Out[_z] = Scalar_product(C3,In);\
}


static void
Set_rot(r32 a, vector n)
{                                               /* Set up a rotation matrix */
    r32                     n11, n22, n33, nxy, nxz, nyz, sina, cosa;

    cosa = cos(a);
    sina = sin(a);

    n11 = n[_x] * n[_x];
    n22 = n[_y] * n[_y];
    n33 = n[_z] * n[_z];

    nxy = n[_x] * n[_y];
    nxz = n[_x] * n[_z];
    nyz = n[_y] * n[_z];

    C1[_x] = n11 + (one - n11) * cosa;
    C1[_y] = nxy * (one - cosa) - n[_z] * sina;
    C1[_z] = nxz * (one - cosa) + n[_y] * sina;

    C2[_x] = nxy * (one - cosa) + n[_z] * sina;
    C2[_y] = n22 + (one - n22) * cosa;
    C2[_z] = nyz * (one - cosa) - n[_x] * sina;

    C3[_x] = nxz * (one - cosa) - n[_y] * sina;
    C3[_y] = nyz * (one - cosa) + n[_x] * sina;
    C3[_z] = n33 + (one - n33) * cosa;
}


static r32
Get_value(u32 *j)
{                                               /* Read a (xx) value from a
                                                 * production string at location j
                                                 * and make it into a real */
    s16                     i = 0;
    r32                     r = 0.0;
    char                    val[40] = "";

    (*j)++;
    (*j)++;

    for (;;) {
        if (Object_Str[*j] == ')')
            break;
        val[i] = Object_Str[*j];
        i++;
        (*j)++;
    }

    val[i] = '\0';
    sscanf(val, "%f", &r);

    if (last_recur)
        r *= fraction;

    return r;
}


// ---------------------------- End of function headers added by Ken ----------------------


static void
L_draw(void)
```

```
{                                           /* Process a production string and
                                             * generate form */
    vector                  pos, end, v, fow, upp, lef;
    u32                     i, max = strlen(Object_Str);
    r32                     r, a, thick_l, ang_l, dis_l, dis2_l, trope_l;
    s16                     vcount, pcount, j;
    char                    temp[max_file], next;
    boolean                 found, poly_on = FALSE;
    u32                     star_num;

 /* Echo production string */
    Get_comline_opt("l", &found, temp);
    if (found) {
        Message("Production     : ");
        for (i = 0; Object_Str[i] != '\0'; i++) {
            Message("%c", Object_Str[i]);
        }
        Message("\n");
    };

 /* Get user form if needed */
    if (user_form)
        Read_form();

 /* Setup vectors */
    pos[_x] = 0.0;
    pos[_y] = 0.0;
    pos[_z] = 0.0;
    fow[_x] = 0.0;
    fow[_y] = 0.0;
    fow[_z] = 1.0;
    lef[_x] = 0.0;
    lef[_y] = 1.0;
    lef[_z] = 0.0;
    upp[_x] = 1.0;
    upp[_y] = 0.0;
    upp[_z] = 0.0;
    Vector_normalize(trope);

 /* Do it */
    Open_datafile();

 /* Start values */
    org.col = col;
    org.dis = dis;
    org.dis2 = dis2;
    org.ang = ang;
    org.thick = thick;
    org.tr = tr;

 /* Feedback */
    Process_start2(strlen(Object_Str));

    for (i = 0; Object_Str[i] != '\0'; i++) {
        Process_update2(i);

        if (polcount > poly_limit)          /* overflow */
            break;

        next = Object_Str[i + 1];             /* the next char in the string */

        switch (Object_Str[i]) {              /* the current char in the string */
          default:
            break;

          case '@':                         /* Marks last recursion level during
                                             * growing phase */
            last_recur = !last_recur;
            if (last_recur) {               /* Store all vars and do fraction */
                thick_l = thick;
                ang_l = ang;
                dis_l = dis;
                dis2_l = dis2;
                trope_l = trope_amount;
                dis *= fraction;
                dis2 *= fraction;
                thick *= fraction;
                ang *= fraction;
```

```
        trope_amount *= fraction;
    } else {                        /* Restore */
        thick = thick_l;
        ang = ang_l;
        dis = dis_l;
        dis2 = dis2_l;
        trope_amount = trope_l;
    }
    break;

  case '+':
    save.ang = ang;
    if (next == '(') {
        ang = ((r32) 0.017453) * Get_value(&i);
        if (last_recur)
            ang *= fraction;
    }
    Set_rot(-ang, upp);
    Util_t(fow, C1, C2, C3, v);
    Vector_copy_r32(v, fow);
    Util_t(lef, C1, C2, C3, v);
    Vector_copy_r32(v, lef);
    Vector_normalize(fow);
    Vector_normalize(lef);
    ang = save.ang;
    break;

  case '-':
    save.ang = ang;
    if (next == '(') {
        ang = ((r32) 0.017453) * Get_value(&i);
        if (last_recur)
            ang *= fraction;
    }
    Set_rot(ang, upp);
    Util_t(fow, C1, C2, C3, v);
    Vector_copy_r32(v, fow);
    Util_t(lef, C1, C2, C3, v);
    Vector_copy_r32(v, lef);
    Vector_normalize(fow);
    Vector_normalize(lef);
    ang = save.ang;
    break;

  case '~':
    if (next == '(')
        r = ((r32) 0.017453) * Get_value(&i);
    else if (rand_set)
        r = ((r32) 0.017453) * rand_amount;
    else
        r = (r32) 6.0;
    a = (Rnd() * r * (r32) 2.0) - r;
    Set_rot(a, upp);
    Util_t(fow, C1, C2, C3, v);
    Vector_copy_r32(v, fow);
    Util_t(lef, C1, C2, C3, v);
    Vector_copy_r32(v, lef);
    Vector_normalize(fow);
    Vector_normalize(lef);
    a = (Rnd() * r * (r32) 2.0) - r;
    Set_rot(a, lef);
    Util_t(fow, C1, C2, C3, v);
    Vector_copy_r32(v, fow);
    Util_t(upp, C1, C2, C3, v);
    Vector_copy_r32(v, upp);
    Vector_normalize(fow);
    Vector_normalize(upp);
    a = (Rnd() * r * (r32) 2.0) - r;
    Set_rot(a, fow);
    Util_t(lef, C1, C2, C3, v);
    Vector_copy_r32(v, lef);
    Util_t(upp, C1, C2, C3, v);
    Vector_copy_r32(v, upp);
    Vector_normalize(lef);
    Vector_normalize(upp);
    break;

  case 't':
```

```
  if ((fow[_x] == (r32) 0.0) && (fow[_y] == (r32) 0.0))
      break;
  save.tr = tr;
  if (trope_set)
      tr = trope_amount;
  if (next == '(') {
      tr = Get_value(&i);
      if (last_recur)
          tr *= fraction;
  }
  Vector_copy_r32(fow, trope);
  trope[_x] = -trope[_x];
  trope[_y] = -trope[_y];
  trope[_z] = (r32) 0.0;
  Vector_normalize(trope);
  r = tr * Scalar_product(fow, trope);
  Set_rot(-r, lef);
  Util_t(fow, C1, C2, C3, v);
  Vector_copy_r32(v, fow);
  Util_t(upp, C1, C2, C3, v);
  Vector_copy_r32(v, upp);
  Vector_normalize(fow);
  Vector_normalize(upp);
  tr = save.tr;
  break;

case '$':
  Vector_min(fow, sky, v);
  if (Vector_length(v) == (r32) 0.0)
      break;
  Vector_product(fow, sky, lef);
  Vector_product(fow, lef, upp);
  if (upp[_z] < (r32) 0.0) {
      upp[_x] = -upp[_x];
      upp[_y] = -upp[_y];
      upp[_z] = -upp[_z];
      lef[_x] = -lef[_x];
      lef[_y] = -lef[_y];
      lef[_z] = -lef[_z];
  }
  break;

case '&':
  save.ang = ang;
  if (next == '(') {
      ang = ((r32) 0.017453) * Get_value(&i);
      if (last_recur)
          ang *= fraction;
  }
  Set_rot(ang, lef);
  Util_t(fow, C1, C2, C3, v);
  Vector_copy_r32(v, fow);
  Util_t(upp, C1, C2, C3, v);
  Vector_copy_r32(v, upp);
  Vector_normalize(fow);
  Vector_normalize(upp);
  ang = save.ang;
  break;

case '^':
  save.ang = ang;
  if (next == '(') {
      ang = ((r32) 0.017453) * Get_value(&i);
      if (last_recur)
          ang *= fraction;
  }
  Set_rot(-ang, lef);
  Util_t(fow, C1, C2, C3, v);
  Vector_copy_r32(v, fow);
  Util_t(upp, C1, C2, C3, v);
  Vector_copy_r32(v, upp);
  Vector_normalize(fow);
  Vector_normalize(upp);
  ang = save.ang;
  break;

case '\\':
  save.ang = ang;
```

```
  if (next == '(') {
      ang = ((r32) 0.017453) * Get_value(&i);
      if (last_recur)
          ang *= fraction;
  }
  Set_rot(-ang, fow);
  Util_t(lef, C1, C2, C3, v);
  Vector_copy_r32(v, lef);
  Util_t(upp, C1, C2, C3, v);
  Vector_copy_r32(v, upp);
  Vector_normalize(lef);
  Vector_normalize(upp);
  ang = save.ang;
  break;

case '/':
  save.ang = ang;
  if (next == '(') {
      ang = ((r32) 0.017453) * Get_value(&i);
      if (last_recur)
          ang *= fraction;
  }
  Set_rot(ang, fow);
  Util_t(lef, C1, C2, C3, v);
  Vector_copy_r32(v, lef);
  Util_t(upp, C1, C2, C3, v);
  Vector_copy_r32(v, upp);
  Vector_normalize(lef);
  Vector_normalize(upp);
  ang = save.ang;
  break;

case '%':
  Set_rot(3.141592654, fow);
  Util_t(lef, C1, C2, C3, v);
  Vector_copy_r32(v, lef);
  Util_t(upp, C1, C2, C3, v);
  Vector_copy_r32(v, upp);
  Vector_normalize(lef);
  Vector_normalize(upp);
  break;

case '|':
  Set_rot(3.141592654, upp);
  Util_t(fow, C1, C2, C3, v);
  Vector_copy_r32(v, fow);
  Util_t(lef, C1, C2, C3, v);
  Vector_copy_r32(v, lef);
  Vector_normalize(fow);
  Vector_normalize(lef);
  break;

case '!':
  if (next == '(') {
      if (last_recur)
          thick *= one + fraction * (Get_value(&i) - one);
      else
          thick *= Get_value(&i);
  } else {
      if (last_recur)
          thick *= one + fraction * ((r32) 0.7 - one);
      else
          thick *= (r32) 0.7;
  }
  break;

case '?':
  if (next == '(') {
      if (last_recur)
          thick *= one + fraction * (Get_value(&i) - one);
      else
          thick *= Get_value(&i);
  } else {
      if (last_recur)
          thick /= one + fraction * ((r32) 0.7 - one);
      else
          thick /= (r32) 0.7;
  }
```

```
  break;

case ':':
  if (next == '(') {
      if (last_recur)
          ang *= one + fraction * (Get_value(&i) - one);
      else
          ang *= Get_value(&i);
  } else {
      if (last_recur)
          ang *= one + fraction * ((r32) 0.9 - one);
      else
          ang *= (r32) 0.9;
  }
  break;

case ';':
  if (next == '(') {
      if (last_recur)
          ang *= one + fraction * (Get_value(&i) - one);
      else
          ang *= Get_value(&i);
  } else {
      if (last_recur)
          ang /= one + fraction * ((r32) 0.9 - one);
      else
          ang /= (r32) 0.9;
  }
  break;

case '\'':
  if (next == '(') {
      r = Get_value(&i);
      if (last_recur) {
          dis *= one + fraction * (r - one);
          dis2 *= one + fraction * (r - one);
      } else {
          dis *= r;
          dis2 *= r;
      }
  } else {
      if (last_recur) {
          dis *= one + fraction * ((r32) 0.9 - one);
          dis2 *= one + fraction * ((r32) 0.9 - one);
      } else {
          dis *= (r32) 0.9;
          dis2 *= (r32) 0.9;
      }
  }
  break;

case '\"':
  if (next == '(') {
      r = Get_value(&i);
      if (last_recur) {
          dis *= one + fraction * (r - one);
          dis2 *= one + fraction * (r - one);
      } else {
          dis *= r;
          dis2 *= r;
      }
  } else {
      if (last_recur) {
          dis /= one + fraction * ((r32) 0.9 - one);
          dis2 /= one + fraction * ((r32) 0.9 - one);
      } else {
          dis /= (r32) 0.9;
          dis2 /= (r32) 0.9;
      }
  }
  break;

case 'Z':
  save.dis2 = dis2;
  if (next == '(') {
      dis2 = Get_value(&i);
      if (last_recur)
          dis2 *= fraction;
```

```
  }
  Vector_plus_fac(pos, fow, dis2, end);
  if (user_form)
      Define_form(pos, end, upp, col);
  else if (closed_form)
      Define_closed(pos, end, upp, col);
  else
      Define_block(pos, end, upp, col);
  Vector_copy_r32(end, pos);
  dis2 = save.dis2;
  break;

case 'F':
  save.dis = dis;
  if (next == '(') {
      dis = Get_value(&i);
      if (last_recur)
          dis *= fraction;
  }
  Vector_plus_fac(pos, fow, dis, end);
  if (user_form)
      Define_form(pos, end, upp, col);
  else if (closed_form)
      Define_closed(pos, end, upp, col);
  else
      Define_block(pos, end, upp, col);
  Vector_copy_r32(end, pos);
  dis = save.dis;
  break;

case '[':
  if (scount > max_stack)
      User_error("Ran out of stack");
  Vector_copy_r32(pos, stack[scount].pos);
  Vector_copy_r32(fow, stack[scount].fow);
  Vector_copy_r32(lef, stack[scount].lef);
  Vector_copy_r32(upp, stack[scount].upp);
  stack[scount].col = col;
  stack[scount].dis = dis;
  stack[scount].dis2 = dis2;
  stack[scount].ang = ang;
  stack[scount].thick = thick;
  stack[scount].tr = tr;
  if (closed_form) {
      Vector_copy_r32(last, stack[scount].last);
      stack[scount].last_col = last_col;
      for (j = 1; j <= 8; j++)
          Vector_copy_r32(last_v[j], stack[scount].last_v[j]);
  }
  scount++;
  break;

case ']':
  scount--;
  Vector_copy_r32(stack[scount].pos, pos);
  Vector_copy_r32(stack[scount].fow, fow);
  Vector_copy_r32(stack[scount].lef, lef);
  Vector_copy_r32(stack[scount].upp, upp);
  col = stack[scount].col;
  dis = stack[scount].dis;
  dis2 = stack[scount].dis2;
  ang = stack[scount].ang;
  thick = stack[scount].thick;
  tr = stack[scount].tr;
  if (closed_form) {
      Vector_copy_r32(stack[scount].last, last);
      last_col = stack[scount].last_col;
      for (j = 1; j <= 8; j++)
          Vector_copy_r32(stack[scount].last_v[j], last_v[j]);
  }
  break;

case '{':
  if (poly_on) {
      pstack[pscount].count = vcount;
      pstack[pscount].ver = (vector *) malloc(vcount * 12L);
      if (pstack[pscount].ver == NULL)
          User_error("Ran out of memory");
```

```
        Vector_copy_max_r32(vcount, ver, pstack[pscount].ver);
        pscount++;
        if (pscount > max_stack)
            User_error("Ran out of stack");
    }
    poly_on = TRUE;
    vcount = (s16) 1;
    pcount = (s16) 1;
    Vector_copy_r32(pos, ver[vcount++]);
    break;

  case 'f':
    save.dis = dis;
    if (next == '(') {
        dis = Get_value(&i);
        if (last_recur)
            dis *= fraction;
    }
    Vector_plus_fac(pos, fow, dis, pos);
    if (poly_on)
        Vector_copy_r32(pos, ver[vcount++]);
    dis = save.dis;
    break;

  case '.':
    if (poly_on)
        Vector_copy_r32(pos, ver[vcount++]);
    break;

  case 'g':
    save.dis = dis;
    if (next == '(') {
        dis = Get_value(&i);
        if (last_recur)
            dis *= fraction;
    }
    Vector_plus_fac(pos, fow, dis, pos);
    dis = save.dis;
    break;

  case 'z':
    save.dis2 = dis2;
    if (next == '(') {
        dis2 = Get_value(&i);
        if (last_recur)
            dis2 *= fraction;
    }
    Vector_plus_fac(pos, fow, dis2, pos);
    if (poly_on)
        Vector_copy_r32(pos, ver[vcount++]);
    dis2 = save.dis2;
    break;

  case '}':
    if (vcount > (s16) 3) {
        for (j = 1; j < vcount - 2; j++) {
            poly_store[pcount][0] = 1;
            poly_store[pcount][1] = j + 1;
            poly_store[pcount][2] = j + 2;
            poly_store[pcount++][3] = j + 2;
        }
        Save_object(vcount - 1, pcount - 1, col);
    }
    poly_on = FALSE;
    if (pscount > 0) {
        pscount--;
        Vector_copy_max_r32(pstack[pscount].count, pstack[pscount].ver, ver);
        vcount = pstack[pscount].count;
        poly_on = TRUE;
    }
    break;

  case 'c':
    if (next == '(') {
        col = (s16) Get_value(&i);
    } else {
        col++;
    }
```

```
            break;
        }
    };

    Process_end2();
}


/* Lparser main ------------------------------------------------------------ */


static void
Help(void)
{
    char                    s[max_file];

    Get_comline_progname(s);
    Message("%s [options] ls-filename\n\n", s);

    Message("%s \tnone      (default) output Lviewer VOL file\n", "vol");
    Message("%s \t-v        output POV object file\n", "pov");
    Message("%s \t-b        output POV blob file\n", "pov");
    Message("%s \t-B        output multiple POV blob files\n", "pov");
    Message("%s \t-c        output inc files instead of pov files\n", "pov");
    Message("%s \t-d        output polyface meshes DXF file\n", "dxf");
    Message("%s \t-3        output 3dfaces DXF file\n", "dxf");
    Message("%s \t-R        output triangles in RAW file\n", "raw");
    Message("%s \t-O        output Blob Sculptor BLB file\n", "blb");
    Message("%s \t-V        output VRML world file\n", "wrl");

    Message("%s \t-X [name] use name.vol as base element\n", "base");
    Message("%s \t-i        link base elements together\n", "base");

    Message("%s \t-S [num]  set string size to num Kbytes\n", "set");
    Message("%s \t-t [num]  set minimum thickness\n", "set");
    Message("%s \t-r [num]  overrule recursion depth\n", "set");
    Message("%s \t-a [num]  overrule angle\n", "set");

    Message("%s \t-u [num]  mutate [num] times\n", "ls");
    Message("%s \t-l        show final L-string\n", "ls");
    Message("%s \t-g        add ground plane\n", "ls");
    Message("%s \t-L [num]  set amount for ~ command\n", "ls");
    Message("%s \t-P [num]  set amount for t command\n", "ls");

    Message("%s \t-p [num]  limit polygons to [num]\n", "limit");

        Message("%s \t-x [num]  multiply x with [num] for campos\n", "new");
        Message("%s \t-y [num]  multiply y with [num] for campos\n", "new");
        Message("%s \t-z [num]  multiply z with [num] for campos\n", "new");
}


// ----------------------------- Function headers added by Ken -----------------------------


void L_Init();

void L_Finish();

void Replace_Once();

void L_System(int N);

void L_Mutate(int N);

void Print_Object();

float Expr_Eval(char *Expr_Str, int First, int Last);

bool BExp_Eval(char *Expr_Str, int First, int Last);


// ------------------------------- Main function (modified by Ken) -------------------------


void main(int argc, char *argv[]) {
  char temp[max_file];
  boolean found;
```

```
// Store the pointers to the comline

s_argc = argc;
s_argv = argv;

// Display header and help file if needed

Message("%s\n", min_bar);
Message("%s v%s (%s)\n", "L-System Parser/Mutator", "5.1", __DATE__);
Message("%s\n", min_bar);

if (argc < 2) {
  Help();
  User_error("Need arguments\n");
};

// Set the option string
strcpy(opts, "VP:OhL:S:iecBbRd3vX:p:t:x:y:z:u:r:a:gl");

// Init files
Init_file_buf(0);
Init_file_buf(1);
Init_file_buf(2);

// Check for all the comline options

Get_comline_opt("t", &found, temp);    // set minimum thickness
if (found)
  sscanf(temp, "%f", &min_thick);

Get_comline_opt("x", &found, temp);    // set x
if (found)
  x = atof(temp);
else
     x = 2;

Get_comline_opt("y", &found, temp);    // set y
if (found)
  y = atof(temp);
else
     y = 2;

Get_comline_opt("z", &found, temp);    // set z
if (found)
  z = atof(temp);
else
     z = 2;

Get_comline_opt("p", &found, temp);    // limit total generated polygons
if (found)
  sscanf(temp, "%ld", &poly_limit);

Get_comline_opt("L", &rand_set, temp); // set random amount
if (rand_set) {
  sscanf(temp, "%f", &rand_amount);
  srand(0);                            /* when using changing amounts of
                                        * randomness for animations we want
                                        * the seed to be the same so we set
                                        * it here */
} else {
  srand(time(NULL));
};

Get_comline_opt("P", &trope_set, temp); // set amount of trope
if (trope_set) {
  sscanf(temp, "%f", &trope_amount);
  srand(0);                             // same for trope animations
};

Get_comline_opt("X", &user_form, x_name);   /* use a vol file as user shape */
Get_comline_opt("i", &closed_form, temp);   /* create closed connected
                                             * cylinders */
Get_comline_opt("d", &dxf1, temp);       /* create a dxf file with inserts */
Get_comline_opt("3", &dxf2, temp);       /* create a dxf with only polygons */
Get_comline_opt("R", &dxf3, temp);       /* create a RAW triangle file */
Get_comline_opt("V", &vrml, temp);       /* create a WRL VRML v1.0 file */
Get_comline_opt("B", &pov_form3, temp);  /* create multiple povray blob files
```

```
                                          * for multi colord blobs */

  Get_comline_opt("O", &blb_form, temp);   /* create a blb blob file */
  Get_comline_opt("b", &pov_form2, temp);  /* create a povray blob file */
  Get_comline_opt("v", &pov_form, temp);   /* create a povray file */
  Get_comline_opt("c", &inc_out, temp);    /* create a povray inc file */

 /*
  * In these cases an user defined form is used. In the other cases a simple
  * block or connected cylinder shape will be used.
  */
  if (pov_form || pov_form2 || pov_form3 || blb_form)
    user_form = TRUE;

  // Read ls file and setup rules

  L_Init();

  L_Mutate(0);

  // Create L-system production string

  L_System((int)recursion);

  // Parse production string and create geometry

  L_draw();

  // Add groundplane

  Get_comline_opt("g", &found, temp);
  if (found)
    Ground_plane();

  Close_datafile();

  // De-allocate the memory allocated for the rules

  L_Finish();

  Message("\n");

////////////////////////////////////////////////////////////////////////////////////////////
////

  // Modified Part of Jan Derer
  // Computation of the best Cameraposition
  // and saving it, into the info.txt file for LV2POVID

  createINFOTXT();

////////////////////////////////////////////////////////////////////////////////////////////
////

}


// --------------------------- Function implementations added by Ken -----------------------


// --------------------------- Main L-system functions -------------------------------------


void L_System(int N) {

        // This function executes the L-system.

        int i;

        Process_start2(lev - 1);

  // Execute N productions in the current L-system

        for (i = 0; i < N; i++) {
                Process_update2(i);
                Replace_Once();
        }
        Message("Size of string : %ld chars\n", strlen(Object_Str));
```

```
        Process_end2();
}

void Replace_Once() {

        // This function executes one production in Object_Str, given the rules.

        long i = 0, New_Len = 0;
        int j, Rule;
        bool Rule_Found;

        New_Str[0] = '\0';    // Clear the new object string

        while (i < Obj_Len) {
                Rule_Found = false;

    // Find the rule with the LONGEST predecessor that applies.

    for (j = 0; j < N_Rules; j++) {
                        if (Rules[j]->Rule_Applies(i)) {
                                if (!Rule_Found) {
                                        Rule = j;
                                        Rule_Found = true;
                                }
                                else {
                                        if (Rules[j]->Prd_Len() >= Rules[Rule]->Prd_Len())
                                        Rule = j;
                                }
                        }
                }

    // If a rule was found, apply it; otherwise, simply add the current
    // character to the new string.

                if (Rule_Found) {
                        i += Rules[Rule]->Do_Replace(i);
                        New_Len = strlen(New_Str);
                }
                else {
                        New_Str[New_Len++] = Object_Str[i++];
                        New_Str[New_Len] = '\0';
                }
        }
        strcpy(Object_Str, New_Str);
        Obj_Len = strlen(Object_Str);
}

void Print_Object() {
  cout << endl << Object_Str << endl;
}


void Replace_Str(char *Search_Str, char *Str1, char *Str2) {

// --------------- Function: Replace one substring with another ------------------
// This function replaces all occurances of Str1 in Search_Str Str2.

        char *Tmp_Ptr = strstr(Search_Str, Str1);  // Find the first occurance of Str1
        char Rem_Str[SUCC_L];
        int i;

        while (Tmp_Ptr != NULL) {
    // Replace this occurance of Str1 with Str2

                i = Tmp_Ptr - Search_Str;
                strcpy(Rem_Str, &Search_Str[i] + strlen(Str1));
                Search_Str[i] = '\0';
                strcat(Search_Str, Str2);
                strcat(Search_Str, Rem_Str);

    // Find the next occurance of Str1

                Tmp_Ptr = strstr(Tmp_Ptr, Str1);
        }
}


// --------------------------- Functions used to initialize rules -------------------------
```

```
void Repl_Var_Addr(char *Search_Str, char *Var, int Addr) {

/* --------------- Function: Replace Variables with Address ----------------------
   This function replaces all occurances of Var in Search_Str with (%Addr).  It is
   used by the constructor of Rule_Type to initialize the rule's successor. */

  char Addr_Str[MAX_NUM_LEN + 3];
  char Num_Str[MAX_NUM_LEN];

  // First, create a new string of the form "(%Addr)"

  strcpy(Addr_Str, "(%");
  strcat(Addr_Str, itoa(Addr, Num_Str, 10));
  strcat(Addr_Str, ")");

  //Replace all occurrances of the variable with its address

  Replace_Str(Search_Str, Var, Addr_Str);

}

void Repl_Vars_WCards(char *Dest, char *Src, char Var_List[MAX_VARS][VAR_L], int *N_Vars) {

/* ------------------ Function: Replace Variables with Wildcards ----------------------
   This function replaces all variables in Src with '#' and stores the resulting string
   in Dest.  (Note: Src is unchanged.)  It also stores the variable names in Var_List,
   beginning with the value of N_Vars.  (This value is incremented each time a variable
   is found and placed in Var_List).  This function is used to initialize a rule's
   predecessor, so that is is easy to see if matches the object string. */

       char *Temp_Str;
       int Var_Len;
       Dest[0] = '\0';

       while (strlen(Src) > 0) {
              Temp_Str = strchr(Src, '(');
              if (Temp_Str == NULL) {      // No more variables found
                     strcat(Dest, Src);
                     Src += strlen(Src);
              }
              else {
                     strncat(Dest, Src, strlen(Src) - strlen(Temp_Str) + 1);

                     // Find variables, separated by commas, until a ')' is found.

                     do {
                            Temp_Str++;
                            Var_Len = strcspn(Temp_Str, ",)");
                            if (Var_Len != 0) {
                                   strcat(Dest, "#");
                                   // Either a ',' or a ')'
                                   strncat(Dest, &Temp_Str[Var_Len], 1);

         // Copy this variable name to Var_List and increment N_Vars.

         strncpy(Var_List[*N_Vars], Temp_Str, Var_Len);
         Var_List[(*N_Vars)++][Var_Len] = '\0';

                                   Temp_Str += Var_Len;
                            }
                     } while (Temp_Str[0] != ')');
                     Src = Temp_Str + 1;
              }
       }
}


Rule_Type::Rule_Type(char *Prd, char *Prv, char *Nxt, char *Cnd, char *Suc, double Prb, int
N_Par) {

/* This function initializes a rule, given all the necessary information about the rule. */

  Prev_Ctx = NULL;
  Next_Ctx = NULL;
  Cond = NULL;
```

```
  // Initialize the predecessor

  strcpy(Pred, Prd);

  // Initialize the first successor (there may be more later)

  Succ[0] = new char[SUCC_L];
  strcpy(Succ[0], Suc);

  // Initialize the probability of applying this successor

  Probs[0] = Prb;
  N_Param = N_Par;

  // Initialize the previous context

  if (strlen(Prv) > 0) {
    Prev_Ctx = new char[PRED_L];
    strcpy(Prev_Ctx, Prv);
  }

  // Initialize the next context

  if (strlen(Nxt) > 0) {
    Next_Ctx = new char[PRED_L];
    strcpy(Next_Ctx, Nxt);
  }

  // Initialize the condition

  if (strlen(Cnd) > 0) {
    Cond = new char[COND_L];
    strcpy(Cond, Cnd);
  }

  N_Succ = 1;  // Initially, there is only one successor.
}

Rule_Type::~Rule_Type() {

/* This function is the rule's destructor.  It de-allocates the memory allocated earlier. */

  for (int i = 0; i < N_Succ; i++)
    delete Succ[i];

  if (Prev_Ctx != NULL) delete Prev_Ctx;
  if (Next_Ctx != NULL) delete Next_Ctx;
  if (Cond != NULL) delete Cond;
}


bool Rule_Type::Same_Rule(char *Prd, char *Prv, char *Nxt, char *Cnd) {

/* This function is used in stochastic L-systems to determine if the rule that was read in
   most recently is really just another possible successor for a rule that was previously
   read.  If the predecessor, previous context, next context, and condition of the new rule
   are identical to this rule, they are parts of the same rule. */

  // First check the predecessor.

  if (strcmp(Prd, Pred) != 0) return false;

  // Check the previous context. Both contexts, and the condition, may be NULL, so we must
  // check for null pointers to avoid errors.

  if (Prev_Ctx != NULL) {
    if (Prv[0] == '\0') return false;
    else if (strcmp(Prv, Prev_Ctx) != 0) return false;
  }
  else if (Prv[0] != '\0') return false;

  // Check the next context.

  if (Next_Ctx != NULL) {
    if (Nxt[0] == '\0') return false;
    else if (strcmp(Nxt, Next_Ctx) != 0) return false;
  }
  else if (Nxt[0] != '\0') return false;
```

```
  // Check the condition.

  if (Cond != NULL) {
    if (Cnd[0] == '\0') return false;
    else if (strcmp(Cnd, Cond) != 0) return false;
  }
  else if (Cnd[0] != '\0') return false;

  // All of the tests pass; these two rules are the same.

  return true;
}


void Rule_Type::Add_Succ(char *Suc, double Prb) {

/* This function is used to add a new possible successor to a rule that has already been
   partially read.  When the function Same_Rule() determines that the rule most recently
   read is really just a new successor for a rule that has already been read, the new
   rule can be inserted with just the successor and probability of using it. */

  // First, create a new successor.

  Succ[N_Succ] = new char[SUCC_L];
  strcpy(Succ[N_Succ], Suc);

  // Set up the probability for this successor.

  Prb += Probs[N_Succ - 1];
  Probs[N_Succ++] = Prb;
}


// ------------------------ Functions used to apply rules --------------------------------


bool Obj_Matches(long i, char *Test_Str, int Dir, int *Block_Len) {

/* ------------------- Function: Does Object_Str match Test_Str? ---------------------------
   This function checks to see if Test_Str matches Object_Str, starting at position i.
   Dir is either 1 or -1, to look forward or backward in Object_Str.  Block_Len is computed;
   it's the length of the block in Object_Str that matches Test_Str.  The character '#' in
   Test_Str is considered any real number (a wildcard). This function returns TRUE if
   Test_Str matches Object_Str at position i; or FALSE otherwise. */

        long Obj_Pos = i;
        long Tst_Pos = (Dir == 1) ? 0 : strlen(Test_Str) - 1;
        char *Obj_Ptr = &Object_Str[Obj_Pos];
        char *Tst_Ptr = &Test_Str[Tst_Pos];

        int Tst_Len = strlen(Test_Str);

        *Block_Len = 0;

        while ((Obj_Pos >= 0) && (Obj_Pos < Obj_Len)) {
               if (*Tst_Ptr != '#') {
                       if (*Tst_Ptr == *Obj_Ptr) {
                              Tst_Pos += Dir;
                              Tst_Ptr += Dir;
                              Obj_Pos += Dir;
                              Obj_Ptr += Dir;
                              *Block_Len = *Block_Len + 1;
                              if ((Tst_Pos < 0) || (Tst_Pos > (Tst_Len - 1)))
                                     return true;
                       }
                       else return false;
               }
               else {
      // We found a '#' in Test_Str. If we are looking at a real number in Object_Str,
      // we must move through Object_Str until we reach the end of that number. Remember,
      // the digits 0..9 or characters '+', '-', 'e', and '.' might be part of a number.

                       if (isdigit(*Obj_Ptr) || (strchr("+-e.", *Obj_Ptr) != NULL)) {
                              Tst_Pos += Dir;
                              Tst_Ptr += Dir;
                              while (isdigit(*Obj_Ptr) || (strchr("+-e.", *Obj_Ptr) != NULL)){
                                     Obj_Pos += Dir;
```

```
                                        Obj_Ptr += Dir;
                                        *Block_Len = *Block_Len + 1;
                                }
                                if ((Tst_Pos < 0) || (Tst_Pos > (Tst_Len - 1)))
                                        return true;
                        }
                        else return false;
                }
        }
        return false;
}


void Find_N_Args(char *Search_Str, float *Arg_List, int N_Args) {

/* ------------------- Function: Find N Arguments -------------------------------------
   This function finds N_Args numbers in Search_Str, and stores them in Arg_List.  The
   numbers must be enclosed in parentheses, and separated by commas. For example, if
   Search_Str == A(1,2), then the arguments returned will be 1 and 2. */

        char Num_Str[MAX_NUM_LEN];
        int i = 0, j;

        while (i < N_Args) {
                Search_Str = strchr(Search_Str, '(');

                while (*Search_Str != ')') {
                        Search_Str++;
                        j = 0;
                        while (isdigit(*(Search_Str + j)) || (strchr("+-e.", *(Search_Str + j))
                                != NULL)) j++;
                        strncpy(Num_Str, Search_Str, j);
                        Num_Str[j] = '\0';
                        Arg_List[i++] = atof(Num_Str);
                        Search_Str += j;
                }
        }
}


void Repl_Addr_Args(char *Search_Str, char *Result_Str, float *Arg_List) {

/* ---------------------- Function: Replace Addresses with Arguments ---------------------
   This function replaces all occurances of (%x) in Search_Str with the value of
   Arg_List[x].  It stores the result in Result_Str.  (Note: Search_Str is unchanged.)
   For example, if Search_Str is A((%0),(%1)) and Arg_List is [1,2], then Result_Str
   will be A(1,2). */

        char *Rpl_Ptr = Search_Str;
        char *Tmp_Ptr = Search_Str;
        char Num_Str[MAX_NUM_LEN];
        Result_Str[0] = '\0';
        int i;

        while (strlen(Rpl_Ptr) > 0) {
                Rpl_Ptr = strchr(Tmp_Ptr, '%');
                if (Rpl_Ptr == NULL) {
                        strcat(Result_Str, Tmp_Ptr);
                        break;
                }
                else if (*(Rpl_Ptr - 1) == '(') {
                        strncat(Result_Str, Tmp_Ptr, strlen(Tmp_Ptr) - strlen(Rpl_Ptr));
                        Rpl_Ptr++;
                        i = 0;
                        while (isdigit(*(Rpl_Ptr + i))) i++;
                        strncpy(Num_Str, Rpl_Ptr, i);
                        Num_Str[i] = '\0';
                        sprintf(Num_Str, "%g\0", Arg_List[atoi(Num_Str)]);
                        strcat(Result_Str, Num_Str);
                        Rpl_Ptr += i;
                        Tmp_Ptr = Rpl_Ptr;
                }
                else Rpl_Ptr++;
        }
}


bool Rule_Type::Rule_Applies(long i) {
```

```
/* ------------------------ Function: Does Rule Apply? -----------------------------
   This function checks to see if a rule applies at position i in Search_Str.  Block_Len
   is calculated; it's the length of characters in Object_Str that must be replaced if this
   rule is applied. */

        int N_Brack, Tmp;
        long j;
  Block_Len = 0; Prev_Len = 0;

  // First, check if the predecessor matches

        if (!Obj_Matches(i, Pred, 1, &Block_Len)) return false;

  // Check for previous context

        if (Prev_Ctx != NULL) {

        // Find the previous context. Don't forget to ignore branches and ignored characters.

                j = i - 1;
                while ((j >= 0) && ((strchr(Ign_Chars, Object_Str[j]) != NULL) ||
                        (Object_Str[j] == '[') || (Object_Str[j] == ']'))) {
                        if (Object_Str[j] == ']') {
                                N_Brack = 0;
                                do {
                                        if (Object_Str[j] == ']') N_Brack++;
                                        if (Object_Str[j] == '[') N_Brack--;
                                        j--;
                                } while (N_Brack > 0);
                        }
                        else j--;
                }
                if (j < 0) return false;
                if (!Obj_Matches(j, Prev_Ctx, -1, &Prev_Len)) return false;
        }

  // Check for next context

        if (Next_Ctx != NULL) {

                // Again, find the next context, making sure to ignore branches.

                j = i + 1;
                while ((j < Obj_Len) && (Object_Str[j] != ']') &&
                        ((strchr(Ign_Chars, Object_Str[j]) != NULL) ||
                        (Object_Str[j] == '['))) {

                        if (Object_Str[j] == '[') {
                                N_Brack = 0;
                                do {
                                        if (Object_Str[j] == '[') N_Brack++;
                                        if (Object_Str[j] == ']') N_Brack--;
                                        j++;
                                } while (N_Brack > 0);
                        }
                        else j++;
                }
                if ((j >= Obj_Len) || (Object_Str[j] == ']')) return false;
                if (!Obj_Matches(j, Next_Ctx, 1, &Tmp)) return false;
        }

  // Check if the condition is met

        return Condition_Met(i);
}


bool Rule_Type::Condition_Met(long i) {

/* ---------------------------- Function: Is Condition Met? ----------------------------
   This function checks to see if a rule's condition is met at position i in Object_Str. */

        char Cond_Str[COND_L];
        float Arg_List[MAX_VARS];

        if (Cond == NULL) return true;
        else {
```

```
                Find_N_Args(&Object_Str[i - Prev_Len], Arg_List, N_Param);
                Repl_Addr_Args(Cond, Cond_Str, Arg_List);
                return BExp_Eval(Cond_Str, 0, strlen(Cond_Str) - 1);
        }
}


int Rule_Type::Do_Replace(long i) {

/* This function executes a rule replacement at position i in Object_Str. */

        char Succ_Str[SUCC_L];
        char Rem_Str[SUCC_L];
        char Num_Str[MAX_NUM_LEN];
        float Arg_List[MAX_VARS];
        char *Suc_Ptr;
        char *Tmp_Ptr;

        int N_Paren;

  double Rnd = (double)(rand() % 100) / 100;
  int j = 0;

  // First, randomly choose the successor we will use.

  while (Probs[j] < Rnd)
    j++;

        // Next, store any necessary numbers in the argument list.

        Find_N_Args(&Object_Str[i - Prev_Len], Arg_List, N_Param);

        // Next, create a successor string with the addresses replaced with numbers.

        Repl_Addr_Args(Succ[j], Succ_Str, Arg_List);

        // Next, evaluate any expressions within the successor string.

        Suc_Ptr = Succ_Str;
        Tmp_Ptr = Succ_Str;

        while (strlen(Suc_Ptr) > 0) {
                Suc_Ptr = strchr(Tmp_Ptr, '(');
                if (Suc_Ptr == NULL) break;
                while (*Suc_Ptr != ')') {
                        Tmp_Ptr = (++Suc_Ptr);
                        N_Paren = 0;
                        while (((*Suc_Ptr != ')') && (*Suc_Ptr != ',')) || (N_Paren != 0)) {
                                if (*Suc_Ptr == '(') N_Paren++;
                                if (*Suc_Ptr == ')') N_Paren--;
                                Suc_Ptr++;
                        }
                        sprintf(Num_Str, "%g\0", Expr_Eval(Tmp_Ptr, 0, Suc_Ptr - Tmp_Ptr - 1));
                        strcpy(Rem_Str, Suc_Ptr);
                        *Tmp_Ptr = '\0';
                        strcat(Succ_Str, Num_Str);
                        strcat(Succ_Str, Rem_Str);
                        Tmp_Ptr += strlen(Num_Str);
                        Suc_Ptr = Tmp_Ptr;
                }
                Suc_Ptr++;
                Tmp_Ptr = Suc_Ptr;
        }

        // Finally, add the replacement string to the end of the new string.

        strcat(New_Str, Succ_Str);

  // Return the length of the section of the object string that was replaced.

  return Block_Len;
}


// ---------------------- Functions used in evaluating expressions --------------------


float Fact_Eval(char *Expr_Str, int First, int Last) {
```

```
        // This function evaluates a factor.

        char Num_Str[MAX_NUM_LEN];
        char *Tmp_Ptr;
        if ((Expr_Str[First] == '(') && (Expr_Str[Last] == ')'))
                return Expr_Eval(Expr_Str, First+1, Last-1);
        else {
                Tmp_Ptr = &Expr_Str[First];
                strncpy(Num_Str, Tmp_Ptr, Last - First + 1);
                 Num_Str[Last - First + 1] = '\0';
                return (float)atof(Num_Str);
        }
}


float Term_Eval(char *Expr_Str, int First, int Last) {

        // This function evaluates a term.

        int N_Paren = 0;
        int i = Last;
        while ((((Expr_Str[i] != '*') && (Expr_Str[i] != '/')) || (N_Paren != 0)) &&
                (i > First)) {
                if (Expr_Str[i] == ')') N_Paren++;
                if (Expr_Str[i] == '(') N_Paren--;
                i--;
        }
        if (Expr_Str[i] == '*')
                return Term_Eval(Expr_Str,First,i-1) * Fact_Eval(Expr_Str,i+1,Last);
        if (Expr_Str[i] == '/')
                return Term_Eval(Expr_Str,First,i-1) / Fact_Eval(Expr_Str,i+1,Last);
        if (i == First)
                return Fact_Eval(Expr_Str,First,Last);
        return 0;
}


float Expr_Eval(char *Expr_Str, int First, int Last) {

        // This function evaluates an arithmetic expression.

        int N_Paren = 0;
        int i = Last;

         while (i > First) {
           if (N_Paren == 0) {
             if ((Expr_Str[i] == '+') && (Expr_Str[i-1] != 'e'))
               return Expr_Eval(Expr_Str,First,i-1) + Term_Eval(Expr_Str,i+1,Last);
             else if ((Expr_Str[i] == '-') && (Expr_Str[i-1] != 'e'))
               return Expr_Eval(Expr_Str,First,i-1) - Term_Eval(Expr_Str,i+1,Last);
           }
           if (Expr_Str[i] == ')') N_Paren++;
           if (Expr_Str[i] == '(') N_Paren--;
           i--;
         }
         return Term_Eval(Expr_Str,First,Last);
}


bool BExp_Eval(char *Expr_Str, int First, int Last) {

        // This function evaluates a boolean expression.

        int N_Paren = 0;
        int i = Last;
        if (Expr_Str[First] == '!')
                return (!BExp_Eval(Expr_Str, First+1, Last));
        else if ((Expr_Str[First] == '(') && (Expr_Str[Last] == ')'))
                return (BExp_Eval(Expr_Str, First+1, Last-1));
        else {
                while (i > First) {
                        if (N_Paren == 0) {
                                if (Expr_Str[i] == '&')
                                        return (BExp_Eval(Expr_Str, First, i-1) &&
                                                BExp_Eval(Expr_Str, i+1, Last));
                                else if (Expr_Str[i] == '|')
                                        return (BExp_Eval(Expr_Str, First, i-1) ||
```

```
                                        BExp_Eval(Expr_Str, i+1, Last));
                                else if (Expr_Str[i] == '=')
                                        return (Expr_Eval(Expr_Str, First, i-1) ==
                                        Expr_Eval(Expr_Str, i+1, Last));
                                else if (Expr_Str[i] == '>') {
                                        if (Expr_Str[i+1] == '=')
                                                return (Expr_Eval(Expr_Str, First, i-1) >=
                                                        Expr_Eval(Expr_Str, i+2, Last));
                                        else
                                                return (Expr_Eval(Expr_Str, First, i-1) >
                                                        Expr_Eval(Expr_Str, i+1, Last));
                                }
                                else if (Expr_Str[i] == '<') {
                                        if (Expr_Str[i+1] == '=')
                                                return (Expr_Eval(Expr_Str, First, i-1) <=
                                                        Expr_Eval(Expr_Str, i+2, Last));
                                        else
                                                return (Expr_Eval(Expr_Str, First, i-1) <
                                                        Expr_Eval(Expr_Str, i+1, Last));
                                }
                        }
      if (Expr_Str[i] == ')') N_Paren++;
      if (Expr_Str[i] == '(') N_Paren--;
                        i--;
                }
       }
       return (Expr_Eval(Expr_Str, First, Last) != 0);
}


// ------------------------ Functions used to initialize the L-system ----------------------


bool Get_Line(char *Str, FILE *In_File) {

/* This function reads a line correctly from the input file. It eliminates any unneeded
   whitespace and ignores the comments (anything beginning with "/*".  It reads until it
   finds a valid line; if no valid lines are found, it returns FALSE. */

  char Tmp_Str[MAX_LIN_LEN];
  int i, j, Len;

  do {

    i = 0; j = 0;

          fgets(Tmp_Str, MAX_LIN_LEN, In_File);
    Len = strlen(Tmp_Str);

    if (Len > MAX_LIN_LEN) return false;

    while (i < Len) {
      // Check for the beginning of a comment

      if (i < (Len - 1)) {
        if ((Tmp_Str[i] == '/') && (Tmp_Str[i+1] == '*'))
          break;
      }

      // If it's a valid character, add it to the string being read

      if (strchr(" \r\n\t", Tmp_Str[i]) == NULL) {
        Str[j] = Tmp_Str[i];
        j++;
      }
      else if ((Tmp_Str[i] == ' ') && (j > 0)) {
        if ((Str[0] == '#') && (Str[j-1] != ' ')) {
          Str[j] = ' ';
          j++;
        }
      }

      i++;
    }

    //Check for trailing space, then terminate the string.

    if ((j > 0) && (Str[j-1] == ' ')) j--;
```

```
    Str[j] = '\0';

  } while ((j == 0) && !feof(In_File));

  return (j > 0);
}


void L_Init(void) {

/* This function reads a .LS file and sets up the initial axiom and rules. It parses the
   file and does some error checking to make sure the rules are valid.
   Note: This function was originally written by Larens Lapre, modified by Ken Kopp. */

  FILE *In_File;
  char name[max_file], Temp_Str[MAX_LIN_LEN];
  boolean found, Read_Ok;
  int i, Len, N_Vars;
  double Prob;

       char *Pred_Ptr, *Succ_Ptr, *Cond_Ptr;
       char *Prev_Ptr, *Next_Ptr;

  char Pred[PRED_L], Succ[SUCC_L], Cond[COND_L];
  char Prev[PRED_L], Next[PRED_L];

  char *CName_Str, *CVal_Str;
  char Num_Str[MAX_NUM_LEN];

  char Var_List[MAX_VARS][VAR_L];

  // Seed the random number generator

  srand(time(NULL));

  // Initialize memory

  stack = (s_rec *) malloc(sizeof(s_rec) * max_stack);
  pstack = (p_rec *) malloc(sizeof(p_rec) * max_stack);
  if ((stack == NULL) || (pstack == NULL))
    User_error("Not enough memory to startup");

  // Get file name

  Get_comline_filename(name);

  if (strstr(name, ".ls") == NULL)
    strcat(name, ".ls");

  In_File = fopen(name, "rt");
  if (!In_File)
    User_error("Cannot find file [%s]", name);

  Message("L-system file  : %s\n", name);

  // Preprocessor statements, such as #define and #ignore:

  strcpy(Ign_Chars, "FfZz+-");

  Get_Line(Temp_Str, In_File);

  while (Temp_Str[0] == '#') {
    if (strstr(Temp_Str, "#define") != NULL) {

      // First, find the name of the constant.

      CName_Str = strchr(Temp_Str, ' ');
      CName_Str++;

      // Next, find the value of the constant. If it is preceded by a '-', put
      // parentheses around it so it won't fool the expression evaluator.

      CVal_Str = strchr(CName_Str, ' ');
      (++CVal_Str)[-1] = '\0';

      if (CVal_Str[0] == '-') {
        Len = strlen(CVal_Str);
        for (i = Len; i > 0; i--)
```

```
        CVal_Str[i] = CVal_Str[i-1];
      CVal_Str[0] = '(';
      CVal_Str[Len+1] = ')';
      CVal_Str[Len+2] = '\0';
    }

    // Insert the constant into the lists of constants.

    strcpy(Const_Names[N_Const], CName_Str);
    strcpy(Const_Vals[N_Const], CVal_Str);
    N_Const++;
    Message("Constant       : %s = %s\n", CName_Str, CVal_Str);
  }
  else if (strstr(Temp_Str, "#ignore:") != NULL) {

    // Find the list of characters that are to be ignored.

    CVal_Str = strchr(Temp_Str, ' ');
    CVal_Str++;
    strcpy(Ign_Chars, CVal_Str);
    Message("Ignored        : %s\n", Ign_Chars);
  }
  Get_Line(Temp_Str, In_File);
}

// Recursion level

sscanf(Temp_Str, "%f", &recursion);

Get_comline_opt("r", &found, Temp_Str);    // Overrule?
if (found)
  sscanf(Temp_Str, "%f", &recursion);
Message("Recursion depth: %g\n", recursion);
lev = (s16) recursion;

fraction = recursion - (r32) lev;
if (fraction > zero) {                 // Check for fraction
  lev++;
  growing = TRUE;
}
else {
  growing = FALSE;
}

// Basic angle

Get_Line(Temp_Str, In_File);

sscanf(Temp_Str, "%f", &ang);

Get_comline_opt("a", &found, Temp_Str);    // Overrule?
if (found)
  sscanf(Temp_Str, "%f", &ang);
Message("Basic angle    : %g\n", ang);
ang = (ang / 180.0) * 3.141592654;

// Thickness

Get_Line(Temp_Str, In_File);

sscanf(Temp_Str, "%f", &thick);
Message("Thickness      : %g\n", thick);
thick /= 100.0;

// Initial Axiom

Get_Line(Temp_Str, In_File);
for (i = 0; i < N_Const; i++)
  Replace_Str(Temp_Str, Const_Names[i], Const_Vals[i]);

strcpy(Object_Str, Temp_Str);
     Obj_Len = strlen(Object_Str);
Message("Axiom          : %s\n", Object_Str);

// Get rules

     while (!feof(In_File)) {
            // First, read the next rule.
```

```
Read_Ok = Get_Line(Temp_Str, In_File);

if (!Read_Ok) break;

           Message("Rule           : %s\n", Temp_Str);

// Execute the preprocessor (#define) statements

for (i = 0; i < N_Const; i++)
  Replace_Str(Temp_Str, Const_Names[i], Const_Vals[i]);

N_Vars = 0;
Prob = 1.0;

// Find the successor part (the section after the last '->').

           Succ_Ptr = strstr(Temp_Str, "->");
           if (Succ_Ptr == NULL) User_error("Error in rule.");

while (strstr(Succ_Ptr + 1, "->") != NULL)
  Succ_Ptr = strstr(Succ_Ptr + 1, "->");

Succ_Ptr += 2;
Succ_Ptr[-2] = '\0';

// Check for stochastic L-system

if (Succ_Ptr[0] == '(') {
  i = 0;
  while (Succ_Ptr[i+1] != ')')
    Num_Str[i++] = Succ_Ptr[i+1];
  Num_Str[i] = '\0';
  Prob = atof(Num_Str);
  Succ_Ptr += (i + 2);
}

if (strlen(Succ_Ptr) == 0) User_error("Error in rule.");

// Find the condition part (the section after the ':').

Cond[0] = '\0';
           Cond_Ptr = strchr(Temp_Str, ':');

if (Cond_Ptr != NULL) {
  (++Cond_Ptr)[-1] = '\0';
  if (strcmp(Cond_Ptr, "*") == 0) Cond_Ptr = NULL;
}

           // Find any 'context sensitivity' blocks before and after the block section.

Next[0] = '\0';
           Next_Ptr = strchr(Temp_Str, '>');

if (Next_Ptr != NULL) {
  (++Next_Ptr)[-1] = '\0';
  if (strcmp(Next_Ptr, "*") == 0) Next_Ptr = NULL;
}

Prev[0] = '\0';
           Prev_Ptr = strchr(Temp_Str, '<');
           if (Prev_Ptr != NULL) {
                  Pred_Ptr = Prev_Ptr + 1;
                  Prev_Ptr = Temp_Str;
                  Pred_Ptr[-1] = '\0';
  if (strcmp(Prev_Ptr, "*") == 0) Prev_Ptr = NULL;
           }
           else Pred_Ptr = Temp_Str;

// Prepare each part of the rule for insertion into the rule list.

if (Prev_Ptr != NULL) Repl_Vars_WCards(Prev, Prev_Ptr, Var_List, &N_Vars);
Repl_Vars_WCards(Pred, Pred_Ptr, Var_List, &N_Vars);
if (Next_Ptr != NULL) Repl_Vars_WCards(Next, Next_Ptr, Var_List, &N_Vars);

if (Cond_Ptr != NULL) {
  strcpy(Cond, Cond_Ptr);
  for (i = 0; i < N_Vars; i++)
    Repl_Var_Addr(Cond, Var_List[i], i);
```

```
    }

    strcpy(Succ, Succ_Ptr);
    for (i = 0; i < N_Vars; i++)
      Repl_Var_Addr(Succ, Var_List[i], i);

               // Insert the rule into the rule list, first checking to see if the predecessor
is already there (in which case,
    // only the new successor needs to be added).

    found = false;
    for (i = 0; i < N_Rules; i++)
      if (Rules[i]->Same_Rule(Pred, Prev, Next, Cond)) {
        Rules[i]->Add_Succ(Succ, Prob);
        found = true;
        break;
      }

    if (!found)
      Rules[N_Rules++] = new Rule_Type(Pred, Prev, Next, Cond, Succ, Prob, N_Vars);
       }

  fclose(In_File);

  // Set start values for F and Z distances

  dis = 100.0;
  dis2 = dis * 0.5;
}


// -------------------------- Functions used to mutate the rules ------------------------


void L_Mutate(int N) {

/* This function executes mutations on the rules that have been initialized. The
   mutations are randomly chosen, based on certain probabilities. It accepts one integer,
   N (the number of mutations to execute). */

  int i, j, k, n1, n2;
  double R;
  char Str1[PRED_L], Str2[PRED_L];

  for (int l = 0; l < N; l++) {

    R = (double)(rand() % 100) / 100;  // Choose a random number between 0 and 1
    i = rand() % N_Rules;              // Choose two random rules
    j = rand() % N_Rules;
    k = 0;

    if (R <= 0.2) {                    // Insert (20% probability)
      n1 = Rules[i]->Get_Pred(Str1);

      // Try up to 10 times to insert the predecessor of rule 1 into another rule.

      while (!Rules[j]->Insert(Str1, n1, false) && (k < 10)) {
        j = rand() % N_Rules;
        k++;
      }
    }
    else if (R <= 0.4) {               // Replace (20% probability)
      n1 = Rules[i]->Get_Pred2(Str1);
      n2 = Rules[j]->Get_Pred2(Str2);

      // Try up to 10 times to replace every occurrance of rule 1's predecessor in
      // rule 1's successor with another string.

      while ((n1 != n2) && (k < 10)) {
        j = rand() % N_Rules;
        n2 = Rules[j]->Get_Pred2(Str2);
        k++;
      }
      if (k < 10) Rules[i]->Replace(Str1, Str2);
    }
    else if (R <= 0.6) {               // Append (20% probability)
      n1 = Rules[i]->Get_Pred(Str1);
```

```
      // Try up to 10 times to append rule 1's predecessor to anoter rule.

      while (!Rules[j]->Insert(Str1, n1, true) && (k < 10)) {
        j = rand() % N_Rules;
        k++;
      }
    }
    else if (R <= 0.8) {               // Swap directions (20% probability)
      Rules[i]->Swap_Dirs();
    }
    else if (R <= 0.9) {               // Swap sizes (10% probability)
      Rules[i]->Swap_Sizes();
    }
    else if (R <= 1.0) {               // Randomize expressions (10% probability)
      Rules[i]->Rand_Expr();
    }

  }
}


int Rule_Type::Get_Pred(char *Str) {

/* This function returns the rule's entire predecessor, including the parameters. */

  int i, N_Vars = 0;
  char Tmp_Str[MAX_NUM_LEN];
  int Len = strlen(Pred);

  Str[0] = '\0';
  for (i = 0; i < Len; i++) {
    if (Pred[i] != '#') {
      Tmp_Str[0] = Pred[i];
      Tmp_Str[1] = '\0';
      strcat(Str, Tmp_Str);
    }
    else {
      strcat(Str, "(%");
      strcat(Str, itoa(N_Vars++, Tmp_Str, 10));
      strcat(Str, ")");
    }
  }
  return N_Param;
}


int Rule_Type::Get_Pred2(char *Str) {

/* This function returns only part of the rule's predecessor--without any parameters.
   For example, if the predecessor is A(x,y), then only A is returned. This is used for
   replacement; if we are replacing B(y,z) with A(x,y), we really want to replace it
   with A(y,z). */

  int i = 0, N_Vars = 0;
  char *Tmp_Str;

  Tmp_Str = strchr(Pred, '(');
  if (Tmp_Str == NULL)
    strcpy(Str, Pred);
  else {
    strncpy(Str, Pred, &Tmp_Str[0] - &Pred[0]);
    while (Tmp_Str[i+1] != ')') {
      if (Tmp_Str[i+1] == '#')
        N_Vars++;
      i++;
    }
  }
  return N_Vars;
}


bool Rule_Type::Insert(char *Str, int N_Args, bool App) {

/* This function inserts Str into a random position in the rule's successor. The position
   must not be inside an expression, so this is checked for. It accepts the string to
   insert, the number of arguments in the string being inserted, and the boolean flag
   App (append if true).  If the number of arguments in the string being inserted is
   greater than the number of parameters in the rule in which it is being inserted, the
```

```
   string will not be inserted and FALSE (indicating failure) will be returned. */

  int i = rand() % N_Succ;
  int j = rand() % strlen(Succ[i]);
  int N_Paren = 0;
  char Rem_Str[SUCC_L];

  if (N_Param < N_Args) return false;

  if (App)
    strcat(Succ[i], Str);
  else {
    for (int k = 0; k < j; k++) {
      if (Succ[i][k] == '(') N_Paren++;
      if (Succ[i][k] == ')') N_Paren--;
    }
    while (N_Paren > 0) {
      if (Succ[i][j] == '(') N_Paren++;
      if (Succ[i][j] == ')') N_Paren--;
      j++;
    }
    strcpy(Rem_Str, &Succ[i][j]);
    Succ[i][j] = '\0';
    strcat(Succ[i], Str);
    strcat(Succ[i], Rem_Str);
  }
  return true;
}


void Rule_Type::Replace(char *Str1, char *Str2) {

/* This function replaces all occurrances of Str1 in a rule's successor with Str2. */

  for (int i = 0; i < N_Succ; i++)
    Replace_Str(Succ[i], Str1, Str2);
}


void Rule_Type::Swap_Dirs() {

/* This function swaps the directions of drawing commands in a rule's successor. It
   makes the following exchanges:
   + <-> -
   & <-> ^
   / <-> \
   | <-> %
   : <-> ;
   ' <-> " */

  int Len;
  for (int i = 0; i < N_Succ; i++) {
    Len = strlen(Succ[i]);
    for (int j = 0; j < Len; j++) {
      if (Succ[i][j] == '+') Succ[i][j] = '-';
      else if (Succ[i][j] == '-') Succ[i][j] = '+';
      else if (Succ[i][j] == '&') Succ[i][j] = '^';
      else if (Succ[i][j] == '^') Succ[i][j] = '&';
      else if (Succ[i][j] == '/') Succ[i][j] = '\\';
      else if (Succ[i][j] == '\\') Succ[i][j] = '/';
      else if (Succ[i][j] == '|') Succ[i][j] = '%';
      else if (Succ[i][j] == '%') {
        if (j == 0) Succ[i][j] = '|';
        else if (Succ[i][j-1] != '(') Succ[i][j] = '|';
      }
      else if (Succ[i][j] == ':') Succ[i][j] = ';';
      else if (Succ[i][j] == ';') Succ[i][j] = ':';
      else if (Succ[i][j] == '\'') Succ[i][j] = '\"';
      else if (Succ[i][j] == '\"') Succ[i][j] = '\'';
    }
  }
}


void Rule_Type::Swap_Sizes() {

/* This function swaps the sizes of drawing commands in a rule's successor. It
   makes the following exchanges:
```

```
   F <-> Z
   f <-> z
   ? <-> ! */

  int Len;
  for (int i = 0; i < N_Succ; i++) {
    Len = strlen(Succ[i]);
    for (int j = 0; j < Len; j++) {
      if (Succ[i][j] == 'F') Succ[i][j] = 'Z';
      else if (Succ[i][j] == 'Z') Succ[i][j] = 'F';
      else if (Succ[i][j] == 'f') Succ[i][j] = 'z';
      else if (Succ[i][j] == 'z') Succ[i][j] = 'f';
      else if (Succ[i][j] == '?') Succ[i][j] = '!';
      else if (Succ[i][j] == '!') Succ[i][j] = '?';
    }
  }
}


void Rule_Type::Rand_Expr() {

/* This function introduces randomness into all the expressions in a rule's successor.
   It multiplies the first term in each expression by a random constant between -2 and 2. */

  int j, N_Paren;
  char Rem_Str[SUCC_L];
  for (int i = 0; i < N_Succ; i++) {
    j = 0;
    while (Succ[i][j] != '\0') {
      if ((Succ[i][j] == '(') || (Succ[i][j] == ',')) {
        strcpy(Rem_Str, &Succ[i][j+1]);
        Succ[i][j+1] = '\0';
        sprintf(&Succ[i][j+1], "(%.2lf)*", (double)(rand() % 100) / 25.0 - 2.0);
        strcat(Succ[i], Rem_Str);

        N_Paren = 0;
        j++;
        while (((Succ[i][j] != ',') && (Succ[i][j] != ')')) || (N_Paren != 0)) {
          if (Succ[i][j] == '(') N_Paren++;
          if (Succ[i][j] == ')') N_Paren--;
          j++;
        }
      }
      else j++;
    }
  }
}


void L_Finish() {
  for (int i = 0; i < N_Rules; i++)
    delete Rules[i];
}

////////////////////////////////////////////////////////////////////////////////////////////
////
/*            Adding functions to computate the best Cameraposition                       */
/*                                    by Jan Derer                                        */

// Absolut-function for variables from type double
double dAbs(double number)
{
       if(number < 0)
              return (-1*number);
       return number;
}

// Computate the percantage "perc" from the Axis x, y, or z
double percOfAxisValues(double perc, char type)
{
       if(perc > 100.0 || perc < 0.0)
              return -1;

       switch(type)
       {
              case 'x':      return (__max(maxX, dAbs(minX)) * (perc/100.00));
                             break;
              case 'y':      return (__max(maxY, dAbs(minY)) * (perc/100.00));
```

```
                                break;
                case 'z':       return (__max(maxZ, dAbs(minZ)) * (perc/100.00));
                                break;
                default :       return -1;
        }
}

// Computate the best cameraposition and create INFO.TXT
void createINFOTXT(void)
{
        char    Center0 = 0;
        double  CamX = 0.0,
                        CamY = 0.0,
                        CamZ = 0.0,
                        CenterX = (maxX+minX)/2,
                        CenterY = (maxY+minY)/2,
                        CenterZ = (maxZ+minZ)/2,
                        maxXYZ = 0.0,
                        maxValue = 0.0;

        printf("\nCalculating the best Cameraposition.\n");

        if(minX >= -1 || (dAbs(minX) <= percOfAxisValues(1.0, 'x')))
                Center0++;
        if(minY >= -1 || (dAbs(minY) <= percOfAxisValues(1.0, 'y')))
                Center0++;
        if(maxZ <= 1 || (maxZ <= percOfAxisValues(1, 'z')))
                Center0++;

        switch(Center0)
        {
                // If the scene is a real 3D-Scene
                case 0 :        CamX = minX*x;
                                CamY = minY*y;
                                CamZ = maxZ*z;
                                break;

                // The scene is 2D and one axis have the expansion near 0
                case 1 :        if(((dAbs(CenterX) <= 1) || (dAbs(CenterX) <=
                                        percOfAxisValues(1.0, 'x'))) &&
                                        ((dAbs(CenterY) <= 1) || (dAbs(CenterY) <=
                                        percOfAxisValues(1.0, 'y'))) &&
                                        ((dAbs(CenterZ) <= 1) || (dAbs(CenterZ) <=
                                        percOfAxisValues(1.0, 'z'))))
                                        {
                                                maxValue = __max(__max(maxX, maxY), maxZ);

                                                if(maxX <= 1)
                                                {
                                                        CamX = maxValue * x;
                                                        CamY = CenterY;
                                                        CamZ = CenterZ;
                                                }
                                                else if(maxY <= 1)
                                                {
                                                        CamX = CenterX;
                                                        CamY = maxValue * y;
                                                        CamZ = CenterZ;
                                                }
                                                else
                                                {
                                                        CamX = CenterX;
                                                        CamY = CenterY;
                                                        CamZ = maxValue * z;
                                                }
                                                break;
                                        }
                                if((dAbs(CenterX) <= 1) || (dAbs(CenterX) <=
                                        percOfAxisValues(1.0, 'x')))
                                {
                                        CamX = (dAbs(CenterY) + dAbs(CenterZ)) * x;
                                        CamY = CenterY;
                                        CamZ = CenterZ;
                                }
                                else if((dAbs(CenterY) <= 1) || (dAbs(CenterY) <=
                                        percOfAxisValues(1.0, 'y')))
                                {
                                        CamX = CenterX;
```

```
                                CamY = (dAbs(CenterX) + dAbs(CenterZ)) * y;
                                CamZ = CenterZ;
                        }
                        else
                        {
                                CamX = CenterX;
                                CamY = CamY;
                                CamZ = (dAbs(CenterX) + dAbs(CenterY)) * z;
                        }
                        break;

        // The scene is 2D and one axis have the expansion near 0 and
        // the expansion of another axis is equal
        case 2 :        if(dAbs(CenterX) > percOfAxisValues(1.0, 'x'))
                                maxXYZ = CenterX;
                        if((dAbs(CenterY) > percOfAxisValues(1.0, 'y')) &&
                                (maxXYZ < CenterY))
                        maxXYZ = CenterY;
                        if((dAbs(CenterZ) > percOfAxisValues(1.0, 'z')) && (maxXYZ <
                                CenterZ))
                                maxXYZ = CenterZ;

                        if((CenterX != maxXYZ) && (__max(maxX,dAbs(minX)) <=
                                percOfAxisValues(1.0, 'x')))
                        {
                                CamX = maxXYZ * x;
                                CamY = CenterY;
                                CamZ = CenterZ;
                        }
                        else if((CenterY != maxXYZ) && (__max(maxY,dAbs(minY)) <=
                                percOfAxisValues(1.0, 'y')))
                        {
                                CamX = CenterX;
                                CamY = maxXYZ * y;
                                CamZ = CenterZ;
                        }
                        else
                        {
                                CamX = CenterX;
                                CamY = CenterY;
                                CamZ = maxXYZ * z;
                        }
                        break;

        // The expansion on every axis is equal and scene is 2D
        default:        maxValue = __max(__max(maxX, maxY), maxZ);

                        if(maxX <= 1)
                        {
                                CamX = maxValue * x;
                                CamY = CenterY;
                                CamZ = CenterZ;
                        }
                        else if(maxY <= 1)
                        {
                                CamX = CenterX;
                                CamY = maxValue * y;
                                CamZ = CenterZ;
                        }
                        else
                        {
                                CamX = CenterX;
                                CamY = CenterY;
                                CamZ = maxValue * z;
                        }
                        break;
}

// Creating INFO.TXT
printf("Writing result into INFO.TXT.\n");

volume_file = fopen("INFO.TXT", "wt");
if (volume_file) {
        fprintf(volume_file, "Model       : (output.vol)\n");
        fprintf(volume_file, "-------------------------------------------------------
                        \n");
        fprintf(volume_file, "Minima xyz  : (%.3f %.3f %.3f)\n", minX, minY, minZ);
        fprintf(volume_file, "Maxima xyz  : (%.3f %.3f %.3f)\n", maxX, maxY, maxZ);
```

```
                fprintf(volume_file, "Center      : (%.4f %.4f %.4f)\n\n", CenterX, CenterY,
                                                                    CenterZ);
                fprintf(volume_file, "Camera\n");
                fprintf(volume_file, "------------------------------------------------------
                                     \n");
                fprintf(volume_file, "Perspective : 11.1364\n");
                fprintf(volume_file, "Scale       : 486.32\n");
                fprintf(volume_file, "Location    : (%.2f %.2f %.2f)\n", CamX, CamY, CamZ);
                fprintf(volume_file, "Looking at  : (%.4f %.4f %.4f)\n", CenterX, CenterY,
                                                                      CenterZ);
                fprintf(volume_file, "Up vector   : (0.0000000 0.0000000 0.0000000)\n\n");
                fprintf(volume_file, "------------------------------------------------------
                                     \n");
                fprintf(volume_file, "Model       : (output.vol)\n");
                fprintf(volume_file, "------------------------------------------------------
                                     \n");
                fprintf(volume_file, "Minima xyz  : (%.3f %.3f %.3f)\n", minX, minY, minZ);
                fprintf(volume_file, "Maxima xyz  : (%.3f %.3f %.3f)\n", maxX, maxY, maxZ);
                fprintf(volume_file, "Center      : (%.4f %.4f %.4f)\n\n", CenterX, CenterY,
                                                                    CenterZ);
                fprintf(volume_file, "Camera\n");
                fprintf(volume_file, "------------------------------------------------------
                                     \n");
                fprintf(volume_file, "Perspective : 11.1364\n");
                fprintf(volume_file, "Scale       : 486.32\n");
                fprintf(volume_file, "Location    : (%.2f %.2f %.2f)\n", CamX, CamY, CamZ);
                fprintf(volume_file, "Looking at  : (%.4f %.4f %.4f)\n", CenterX, CenterY,
                                                                      CenterZ);
                fprintf(volume_file, "Up vector   : (0.0000000 0.0000000 0.0000000)\n\n");
                fprintf(volume_file, "------------------------------------------------------
                                     \n\n");
        }
        else
                User_error("Cannot open file INFO.TXT\n");

        fclose(volume_file);

        printf("Finish.\n\n");
}

////////////////////////////////////////////////////////////////////////////////////////////
////

/* ------------------------------------------------------------------------
   End of file.
   ------------------------------------------------------------------------
*/
```

Quelltexte von Visual L:

Dateiname: VisualL.dpr

```
program VisualL;

{%ToDo 'VisualL.todo'}

uses
  Forms,
  MDIMainSource in 'MDIMainSource.pas' {MDIMain},
  MDIChildSource in 'MDIChildSource.pas' {MDIChildMain},
  AboutSource in 'AboutSource.pas' {AboutForm},
  CFGSource in 'CFGSource.pas' {CFGForm},
  ShowPicSource in 'ShowPicSource.pas' {ShowPicForm},
  Direct3DSource in 'Direct3DSource.pas' {Direct3DForm},
  TurtleComSource in 'TurtleComSource.pas' {TurtleComForm},
  CondFormSource in 'CondFormSource.pas' {CondForm},
  ConsoleFormSource in 'ConsoleFormSource.pas' {ConsoleFom};

{$R *.res}

begin
  Application.Initialize;
  Application.Title := 'Visual L Version 1.0';
  Application.CreateForm(TMDIMain, MDIMain);
  Application.CreateForm(TConsoleFom, ConsoleFom);
```

```
  Application.Run;
end.
```

Dateiname: AboutSource.pas

```
{ ---------------------------------------------------------------------------

  Autor:                Jan Derer
  Datum:                05. 06. 04
  Kontakt:              Darkhabits@aol.com
  Programmname:         VisualL
  Version:              1.0

  Klassenname:          TAboutForm
  Version:              1.0
  Kurzbeschreibung:
  Diese Klasse repräsentiert das Info-Fenster.

  --------------------------------------------------------------------------- }

unit AboutSource;

{ ---------------------------------------------------------------------------
                BESCHREIBUNG DER SCHNITTSTELLE DER UNIT
  --------------------------------------------------------------------------- }

interface

{ ---------------------------------------------------------------------------
                Liste alle öffentlich eingebundenen Units
  --------------------------------------------------------------------------- }

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, jpeg, ExtCtrls;

{ ---------------------------------------------------------------------------
                    Deklaration eigener Datentypen
  --------------------------------------------------------------------------- }

type

{ ---------------------------------------------------------------------------
                  Klassenbeschreibung für TAboutForm
  --------------------------------------------------------------------------- }

  TAboutForm = class(TForm)

    // Auflistung aller eingebundenen Komponenten der Klasse
    Image1: TImage;
    Panel1: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    BitBtn1: TBitBtn;
    Label4: TLabel;

    // Auflistung aller Methoden für die Ereignisverarbeitung
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

{ ---------------------------------------------------------------------------
                        Öffentliche globale Variablen
  --------------------------------------------------------------------------- }

var
  AboutForm: TAboutForm;        // Die Instanz von TAboutForm ist öffentlich für alle anderen
                                // Units erreichbar

{ ---------------------------------------------------------------------------
                        IMPLEMENTATIONSTEIL DER UNIT
  --------------------------------------------------------------------------- }
```

```
implementation

{ ------------------------------------------------------------------------
                              Compiler-Schalter
  ------------------------------------------------------------------------ }

{$R *.dfm}

{ ------------------------------------------------------------------------
                  Liste aller private eingebundenen Units
  ------------------------------------------------------------------------ }


{ ------------------------------------------------------------------------

  Methodenname:         FormClose
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var TCloseAction
        Name:           Action
        Bedeutung:      Gibt an, wie das Fenster geschlossen werden ssoll

  Methodenbeschreibung:
  Methode wird beim Beenden des Formulars aufgerufen.

  ------------------------------------------------------------------------ }

procedure TAboutForm.FormClose(Sender: TObject; var Action: TCloseAction);

begin
 Action := caFree;      // Speicher freigeben
end;


{ ------------------------------------------------------------------------

  Methodenname:         BitBtn1Click
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Schließt das Formular, wenn auf "Schließen" gelickt wird.

  ------------------------------------------------------------------------ }

procedure TAboutForm.BitBtn1Click(Sender: TObject);

begin
 Close;         // Schließe das Formular
end;

end.
```

Dateiname: CFGSource.pas

```
{ ------------------------------------------------------------------------

  Autor:                Jan Derer
  Datum:                05. 06. 04
  Kontakt:              Darkhabits@aol.com
  Programmname:         VisualL
  Version:              1.0

  Klassenname:          TCFGForm
  Version:              1.0
  Kurzbeschreibung:
  Diese Klasse repräsentiert das CFG-Formular.

  ------------------------------------------------------------------------ }

unit CFGSource;
```

```
{ ---------------------------------------------------------------------------
                BESCHREIBUNG DER SCHNITTSTELLE DER UNIT
  --------------------------------------------------------------------------- }

interface

{ ---------------------------------------------------------------------------
                Liste alle öffentlich eingebundenen Units
  --------------------------------------------------------------------------- }

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, Buttons, ExtCtrls;

{ ---------------------------------------------------------------------------
                    Deklaration eigener Datentypen
  --------------------------------------------------------------------------- }

type

{ ---------------------------------------------------------------------------
                  Klassenbeschreibung für TCFGForm
  --------------------------------------------------------------------------- }

  TCFGForm = class(TForm)

    // Auflistung aller eingebundenen Komponenten der Klasse
    Panel2: TPanel;
    PageControl1: TPageControl;
    OptionsSheet: TTabSheet;
    LightCamSheet: TTabSheet;
    GroupBox3: TGroupBox;
    Label12: TLabel;
    Label13: TLabel;
    Label14: TLabel;
    Label15: TLabel;
    AACheck: TCheckBox;
    TargaCheck: TCheckBox;
    WidthEdit: TEdit;
    HeightEdit: TEdit;
    PathPOVEdit: TEdit;
    PathQPOVEdit: TEdit;
    SearchPOVBtn: TButton;
    SearchQPOVBtn: TButton;
    Panel1: TPanel;
    Panel3: TPanel;
    GroupBox1: TGroupBox;
    CamXTrack: TTrackBar;
    CamYTrack: TTrackBar;
    CamZTrack: TTrackBar;
    GroupBox2: TGroupBox;
    LightXTrack: TTrackBar;
    LightYTrack: TTrackBar;
    LightZTrack: TTrackBar;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    OKBtn: TBitBtn;
    CancelBtn: TBitBtn;
    CamXLabel: TLabel;
    CamYLabel: TLabel;
    CamZLabel: TLabel;
    LightXLabel: TLabel;
    LightYLabel: TLabel;
    LightZLabel: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Direct3DBtn: TButton;
    Splitter1: TSplitter;

    // Auflistung aller Methoden für die Ereignisverarbeitung
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure CancelBtnClick(Sender: TObject);
    procedure OKBtnClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
```

```
    procedure CamXTrackChange(Sender: TObject);
    procedure CamYTrackChange(Sender: TObject);
    procedure CamZTrackChange(Sender: TObject);
    procedure LightXTrackChange(Sender: TObject);
    procedure LightYTrackChange(Sender: TObject);
    procedure LightZTrackChange(Sender: TObject);
    procedure SearchPOVBtnClick(Sender: TObject);
    procedure WidthEditKeyPress(Sender: TObject; var Key: Char);
    procedure SearchQPOVBtnClick(Sender: TObject);
    procedure Direct3DBtnClick(Sender: TObject);

  // Auflistung aller privaten Methoden als Prototypen
  private
    function convertRealForTracker(value : real48) : integer;
  public
    { Public-Deklarationen }
  end;

{ ---------------------------------------------------------------------------
                          Öffentliche globale Variablen
  --------------------------------------------------------------------------- }

var
  CFGForm: TCFGForm;    // Die Instanz von TCFGForm ist öffentlich für alle anderen
                        // Units erreichbar

{ ---------------------------------------------------------------------------
                          IMPLEMENTATIONSTEIL DER UNIT
  --------------------------------------------------------------------------- }

implementation

{ ---------------------------------------------------------------------------
                              Compiler-Schalter
  --------------------------------------------------------------------------- }

{$R *.dfm}

{ ---------------------------------------------------------------------------
                  Liste aller private eingebundenen Units
  --------------------------------------------------------------------------- }

uses MDIMainSource, imagehlp, Direct3DSource;

{ ---------------------------------------------------------------------------
           Globale Variablen die nur in dieser Unit erreichbar sind
  --------------------------------------------------------------------------- }

var     CamX : real48;  // Enthält den Wert für die x-Achse für die Kamera
        CamY : real48;  // Enthält den Wert für die x-Achse für die Kamera
        CamZ : real48;  // Enthält den Wert für die x-Achse für die Kamera

{ ---------------------------------------------------------------------------

  Methodenname:         convertRealForTracker
  Rückgabewert
        Typ:            integer
        Bedeutung:      Gibt eine Ganzzahl für die Tracker-Komponente zurück
  Parameter
        Typ:            real48
        Name:           value
        Bedeutung:      Fließkommazahl aus der CFG-Datei

  Methodenbeschreibung:
  Rechnet eine Fließkommazahl aus der CFG-Datei um, in eine ganze Zahl für die
  Tracker-Komponente. Dabei ist die Auflösung 0.5 und geht von +10 bis -10.

  --------------------------------------------------------------------------- }

function TCFGForm.convertRealForTracker(value: real48) : integer;

begin

 // Wenn die Zahl größer 10 ist, dann weise das Maximum zu und beende
 if value > 10 then
  begin
   result := 20;
   Exit;
```

```
  end

 // Wenn die Zahl kleiner -10 ist, dann weise das Minimum zu und beende
 else if value < -10 then
  begin
   result := -20;
   Exit;
  end

 // Wenn die Zahl gleich 0 ist, dann weise die 0 zu und beende
 else if value = 0 then
  begin
   result := 0;
   Exit;
  end;

 // Multipliziere die Fließkommazahl mit zwei und runde den Wert auf, bzw. ab.
 result := round(value * 2);

end;


{ ------------------------------------------------------------------------------

  Methodenname:         FormClose
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var TCloseAction
        Name:           Action
        Bedeutung:      Gibt an, wie das Fenster geschlossen werden ssoll

  Methodenbeschreibung:
  Methode wird beim Beenden des Formulars aufgerufen.

  ------------------------------------------------------------------------------ }

procedure TCFGForm.FormClose(Sender: TObject; var Action: TCloseAction);

begin
 if D3DActive then      // Falls ein Direct3D-Fenster offen ist, dann schließe es
  D3DForm.Close;
 Action := caFree;      // Speicher freigeben
end;


{ ------------------------------------------------------------------------------

  Methodenname:         CancelBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Diese Methode wird aufgerufen, wenn auf "Abbrechen" geklickt wird. Das
  Formular wird geschlossen

  ------------------------------------------------------------------------------ }

procedure TCFGForm.CancelBtnClick(Sender: TObject);
begin
 Info.CamX := CamX;     // Übernehme die Werte der Kamera
 Info.CamY := CamY;
 Info.CamZ := CamZ;
 Close;                 // Schließe das Fenster
end;


{ ------------------------------------------------------------------------------

  Methodenname:         OKBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
```

```
  Methodenbeschreibung:
  Diese Methode wird aufgerufen, wenn auf "Ok" geklickt wird. Das
  Formular wird geschlossen und alle Werte übernommen.

  ------------------------------------------------------------------------- }

procedure TCFGForm.OKBtnClick(Sender: TObject);

begin

 // Übergebe alle Werte an die Info-Struktur
 Info.AA := AACheck.Checked;
 Info.Bitmap := not TargaCheck.Checked;
 Info.Width := StrToInt(WidthEdit.Text);
 Info.Height := StrToInt(HeightEdit.Text);
 Info.PathPOV := PathPOVEdit.Text;
 Info.PathQPOV := PathQPOVEdit.Text;
 Info.CamX := StrToFloat(CamXLabel.Caption);
 Info.CamY := StrToFloat(CamYLabel.Caption);
 Info.CamZ := StrToFloat(CamZLabel.Caption);
 Info.LightX := StrToFloat(LightXLabel.Caption);
 Info.LightY := StrToFloat(LightYLabel.Caption);
 Info.LightZ := StrToFloat(LightZLabel.Caption);

 Close;         // Schließe das Formular
end;


{ -------------------------------------------------------------------------

  Methodenname:         FormShow
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird ausgeführt, wenn das Fenster sichtbar wird.

  ------------------------------------------------------------------------- }

procedure TCFGForm.FormShow(Sender: TObject);

begin

 // Alle Werte aus der Info-Struktur an die Komponenten übergeben
 AACheck.Checked := Info.AA;
 TargaCheck.Checked := not Info.Bitmap;
 WidthEdit.Text := IntToStr(Info.Width);
 HeightEdit.Text := IntToStr(Info.Height);
 PathPOVEdit.Text := Info.PathPOV;
 PathQPOVEdit.Text := Info.PathQPOV;

 // Kamera und Licht initialisieren
 CamX := Info.CamX;
 CamY := Info.CamY;
 CamZ := Info.CamZ;
 CamXTrack.Position := convertRealForTracker(Info.CamX);
 CamXLabel.Caption := FloatToStr(CamXTrack.Position * 0.5);
 CamYTrack.Position := convertRealForTracker(Info.CamY);
 CamYLabel.Caption := FloatToStr(CamYTrack.Position * 0.5);
 CamZTrack.Position := convertRealForTracker(Info.CamZ);
 CamZLabel.Caption := FloatToStr(CamZTrack.Position * 0.5);
 LightXTrack.Position := convertRealForTracker(Info.LightX);
 LightXLabel.Caption := FloatToStr(LightXTrack.Position * 0.5);
 LightYTrack.Position := convertRealForTracker(Info.LightY);
 LightYLabel.Caption := FloatToStr(LightYTrack.Position * 0.5);
 LightZTrack.Position := convertRealForTracker(Info.LightZ);
 LightZLabel.Caption := FloatToStr(LightZTrack.Position * 0.5);
end;


{ -------------------------------------------------------------------------

  Methodenname:         CamXTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
```

```
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  ------------------------------------------------------------------------- }

procedure TCFGForm.CamXTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if CamXTrack.Position = 0 then
  begin
   CamXLabel.Caption := '0';
   Info.CamX := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   CamXLabel.Caption := FloatToStrF((CamXTrack.Position * 0.5), ffFixed, 1, 1);
   Info.CamX := CamXTrack.Position * 0.5;
  end;
end;


{ -------------------------------------------------------------------------

  Methodenname:         CamYTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  ------------------------------------------------------------------------- }

procedure TCFGForm.CamYTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if CamYTrack.Position = 0 then
  begin
   CamYLabel.Caption := '0';
   Info.CamY := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   CamYLabel.Caption := FloatToStrF((CamYTrack.Position * 0.5), ffFixed, 1, 1);
   Info.CamY := CamYTrack.Position * 0.5;
  end;
end;


{ -------------------------------------------------------------------------

  Methodenname:         CamZTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  ------------------------------------------------------------------------- }

procedure TCFGForm.CamZTrackChange(Sender: TObject);
```

```
begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if CamZTrack.Position = 0 then
  begin
   CamZLabel.Caption := '0';
   Info.CamZ := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   CamZLabel.Caption := FloatToStrF((CamZTrack.Position * 0.5), ffFixed, 1, 1);
   Info.CamZ := CamZTrack.Position * 0.5;
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         LightXTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  --------------------------------------------------------------------------- }

procedure TCFGForm.LightXTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if LightXTrack.Position = 0 then
  begin
   LightXLabel.Caption := '0';
   Info.LightX := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   LightXLabel.Caption := FloatToStrF((LightXTrack.Position * 0.5), ffFixed, 1, 1);
   Info.LightX := LightXTrack.Position * 0.5;
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         LightYTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  --------------------------------------------------------------------------- }

procedure TCFGForm.LightYTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if LightYTrack.Position = 0 then
  begin
   LightYLabel.Caption := '0';
   Info.LightY := 0;
  end

 // Ansonsten berechne den neuen Wert
```

```
 else
  begin
   LightYLabel.Caption := FloatToStrF((LightYTrack.Position * 0.5), ffFixed, 1, 1);
   Info.LightY := LightYTrack.Position * 0.5;
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         LightZTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  --------------------------------------------------------------------------- }

procedure TCFGForm.LightZTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if LightZTrack.Position = 0 then
  begin
   LightZLabel.Caption := '0';
   Info.LightZ := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   LightZLabel.Caption := FloatToStrF((LightZTrack.Position * 0.5), ffFixed, 1, 1);
   Info.LightZ := LightZTrack.Position * 0.5;
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         SearchPOVBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Diese Methode sucht nach den Pfad von POV-Ray, wenn auf den entsprechenden
  Button geklickt wurde.

  --------------------------------------------------------------------------- }

procedure TCFGForm.SearchPOVBtnClick(Sender: TObject);

var temp : array[0..255] of char;       // Speichert den Pfad von POV-Ray

begin
 SearchTreeForFile('C:\', 'pvengine.exe', temp); // Suche den kompletten Baum ab
 PathPOVEdit.Text := ExtractFileDir(temp);       // Extrahiere den keinen Pfad
end;


{ ---------------------------------------------------------------------------

  Methodenname:         RecursEditKeyPress
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var char
        Name:           Key
        Bedeutung:      Zeiger auf das Zeichen, das eingegeben wurde

  Methodenbeschreibung:
```

```
  Bei jeder Zeicheneingabe für Editierfelder die nur Zahlen akzeptieren,
  wird diese Methode aufgerufen.

  ------------------------------------------------------------------------- }

procedure TCFGForm.WidthEditKeyPress(Sender: TObject; var Key: Char);

begin

 // Lasse nur Zahlen und den Rücklauf als Eingabe zu
 if (Key in ['0'..'9']) or (Key = #8) then
  Key := Key
 else
  Key := #0;
end;


{ -------------------------------------------------------------------------

  Methodenname:         SearchQPOVBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Diese Methode sucht nach den Pfad von QuietPOV, wenn auf den entsprechenden
  Button geklickt wurde.

  ------------------------------------------------------------------------- }

procedure TCFGForm.SearchQPOVBtnClick(Sender: TObject);

var temp : array[0..255] of char;       // Speichert den Pfad von POV-Ray

begin
 SearchTreeForFile('C:\', 'QuietPOV.exe', temp); // Suche den kompletten Baum ab
 PathQPOVEdit.Text := ExtractFileDir(temp);      // Extrahiere den keinen Pfad
end;


{ -------------------------------------------------------------------------

  Methodenname:         Direct3DBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Die Methode wird ausgeführt, wenn auf den Button "3D-Ansicht" geklickt wird.

  ------------------------------------------------------------------------- }

procedure TCFGForm.Direct3DBtnClick(Sender: TObject);

begin

 // Erzeuge ein Instanz des Direct3D-Fensters, wenn noch keine existiert
 if D3DActive = false then
  begin
   D3DForm := TDirect3DForm.Create(Application);
   D3DActive := true;
  end;
end;

end.
```

Dateiname: CondFormSource.pas

```
{ -------------------------------------------------------------------------

  Autor:                Jan Derer
  Datum:                05. 06. 04
  Kontakt:              Darkhabits@aol.com
  Programmname:         VisualL
  Version:              1.0
```

```
  Klassenname:          TCondForm
  Version:              1.0
  Kurzbeschreibung:
  Diese Klasse repräsentiert das Fenster, das zum Anzeigen der Ausdrücke für
  den Bedingungsteil benutzt wird.

  ------------------------------------------------------------------------- }

unit CondFormSource;

{ -------------------------------------------------------------------------
                BESCHREIBUNG DER SCHNITTSTELLE DER UNIT
  ------------------------------------------------------------------------- }

interface

{ -------------------------------------------------------------------------
                Liste alle öffentlich eingebundenen Units
  ------------------------------------------------------------------------- }

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons;

{ -------------------------------------------------------------------------
                     Deklaration eigener Datentypen
  ------------------------------------------------------------------------- }

type

{ -------------------------------------------------------------------------
                   Klassenbeschreibung für TCondForm
  ------------------------------------------------------------------------- }

  TCondForm = class(TForm)

    // Auflistung aller eingebundenen Komponenten der Klasse
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    BitBtn1: TBitBtn;

    // Auflistung aller Methoden für die Ereignisverarbeitung
    procedure BitBtn1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

{ -------------------------------------------------------------------------
                         Öffentliche globale Variablen
  ------------------------------------------------------------------------- }

var
  CondForm: TCondForm;  // Die Instanz von TCondForm ist öffentlich für alle anderen
                        // Units erreichbar

{ -------------------------------------------------------------------------
                         IMPLEMENTATIONSTEIL DER UNIT
  ------------------------------------------------------------------------- }

implementation

{ -------------------------------------------------------------------------
                             Compiler-Schalter
  ------------------------------------------------------------------------- }
```

```
{$R *.dfm}

{ ----------------------------------------------------------------------------
                    Liste aller private eingebundenen Units
  ---------------------------------------------------------------------------- }

uses MDIMainSource;


{ ----------------------------------------------------------------------------

  Methodenname:         BitBtn1Click
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Schließt das Formular, wenn auf "Schließen" gelickt wird.

  ---------------------------------------------------------------------------- }

procedure TCondForm.BitBtn1Click(Sender: TObject);

begin
 Close;         // Schließe das Fenster
end;


{ ----------------------------------------------------------------------------

  Methodenname:         FormClose
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var TCloseAction
        Name:           Action
        Bedeutung:      Gibt an, wie das Fenster geschlossen werden ssoll

  Methodenbeschreibung:
  Methode wird beim Beenden des Formulars aufgerufen.

  ---------------------------------------------------------------------------- }

procedure TCondForm.FormClose(Sender: TObject; var Action: TCloseAction);

begin
 CondActive := false;   // Signalisiert, dass keine Instanz mehr gibt
 Action := caFree;      // Speicher freigeben
end;

end.
```

Dateiname: ConsoleFormSource.pas

```
{ ----------------------------------------------------------------------------

  Autor:                Jan Derer
  Datum:                05. 06. 04
  Kontakt:              Darkhabits@aol.com
  Programmname:         VisualL
  Version:              1.0

  Klassenname:          TConsoleFom
  Version:              1.0
  Kurzbeschreibung:
  Diese Klasse repräsentiert das Fenster, das zum Anzeigen der Konsolenausgabe
  benutzt wird.

  ---------------------------------------------------------------------------- }

unit ConsoleFormSource;

{ ----------------------------------------------------------------------------
                  BESCHREIBUNG DER SCHNITTSTELLE DER UNIT
```

```
  ------------------------------------------------------------------------- }

interface

{ -------------------------------------------------------------------------
                Liste alle öffentlich eingebundenen Units
  ------------------------------------------------------------------------- }

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons, ExtCtrls;

{ -------------------------------------------------------------------------
                    Deklaration eigener Datentypen
  ------------------------------------------------------------------------- }

type

{ -------------------------------------------------------------------------
                   Klassenbeschreibung für TConsoleFom
  ------------------------------------------------------------------------- }

  TConsoleFom = class(TForm)

    // Auflistung aller eingebundenen Komponenten der Klasse
    Panel1: TPanel;
    Panel2: TPanel;
    CloseBtn: TBitBtn;
    ListBox1: TListBox;

    // Auflistung aller Methoden für die Ereignisverarbeitung
    procedure CloseBtnClick(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

{ -------------------------------------------------------------------------
                         Öffentliche globale Variablen
  ------------------------------------------------------------------------- }

var
  ConsoleFom: TConsoleFom;      // Die Instanz von TConsoleFom ist öffentlich für alle anderen
                                // Units erreichbar

{ -------------------------------------------------------------------------
                         IMPLEMENTATIONSTEIL DER UNIT
  ------------------------------------------------------------------------- }

implementation

{ -------------------------------------------------------------------------
                              Compiler-Schalter
  ------------------------------------------------------------------------- }

{$R *.dfm}


{ -------------------------------------------------------------------------

  Methodenname:         CloseBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Schließt das Formular, wenn auf "Schließen" gelickt wird.

  ------------------------------------------------------------------------- }

procedure TConsoleFom.CloseBtnClick(Sender: TObject);

begin
 Close;         // Formular schließen
end;
```

```
end.
```

Dateiname: Direct3DSource.pas

```
{ ------------------------------------------------------------------------

  Autor:                Jan Derer
  Datum:                05. 06. 04
  Kontakt:              Darkhabits@aol.com
  Programmname:         VisualL
  Version:              1.0

  Klassenname:          TDirect3DForm
  Version:              1.0
  Kurzbeschreibung:
  Diese Klasse repräsentiert das Fenster zur Direct3D-Anzeige.

  ------------------------------------------------------------------------ }

unit Direct3DSource;

{ ------------------------------------------------------------------------
                BESCHREIBUNG DER SCHNITTSTELLE DER UNIT
  ------------------------------------------------------------------------ }

interface

{ ------------------------------------------------------------------------
                Liste alle öffentlich eingebundenen Units
  ------------------------------------------------------------------------ }

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Direct3D8, d3dx8;

{ ------------------------------------------------------------------------
                    Deklaration eigener Datentypen
  ------------------------------------------------------------------------ }

type

  // Datenstruktur zum Speichern eines Vertex-Punktes
  TMyVertex = record
    x,y,z      : single;   // Position des Vertex
    nx, ny, nz : single;   // Normalen des Vertex
    color      : dword;    // Farbe des Vertex
    end;


  // Deklaration der Vertex-Liste für 36 Vertex-Punkte
  TMyVertices = array [0..35] of TMyVertex;

{ ------------------------------------------------------------------------
                  Klassenbeschreibung für TDirect3DForm
  ------------------------------------------------------------------------ }

  TDirect3DForm = class(TForm)

    // Auflistung aller Methoden für die Ereignisverarbeitung
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCreate(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);

  // Auflistung aller privaten Methoden als Prototypen und privaten Attribute
  private

    // Private Attribute
    lpd3d       : IDIRECT3D8;                  // Handle für die Direct3D Kommunikation
    lpd3ddevice : IDirect3DDevice8;            // Handle für ein Direct3D Device
    MyVB        : IDirect3DVertexBuffer8;      // Attribut für den Vertex-Buffer
    light       : D3DLight8;                   // Attribut fürs Licht
    material    : D3DMATERIAL8;                // Attribut fürs Material

    // Private Methoden
    procedure FatalError(hr : HResult; FehlerMsg : string);
```

```
    procedure D3DInit;
    procedure D3DShutdown;
    procedure D3DInitScene;
    procedure D3DKillScene;
    procedure D3DRender;
    procedure MyIdleHandler (Sender: TObject; var Done: Boolean);
  public
    { Public-Deklarationen }
  end;

{ ---------------------------------------------------------------------------
        Deklaration von Konstanten und initalisieren von Variablen
  --------------------------------------------------------------------------- }

const

  // Definieren wie die FVF auszusehen hat
  D3D8T_CUSTOMVERTEX = ( D3DFVF_XYZ or D3DFVF_NORMAL or D3DFVF_DIFFUSE );

  // Alle 36 Vertex-Punkte für einen Kubus
  MyVertices : TMyVertices = (
  // Würfel                                                          // Farbe rot
    (x :-1.0; y :-1.0; z :-1.0; nx: 0.0; ny: 0.0; nz: -1.0; color : $FFF00000 ),  // Vorn
    (x :-1.0; y : 1.0; z :-1.0; nx: 0.0; ny: 0.0; nz: -1.0; color : $FFF00000 ),
    (x : 1.0; y : 1.0; z :-1.0; nx: 0.0; ny: 0.0; nz: -1.0; color : $FFF00000 ),
    (x : 1.0; y : 1.0; z :-1.0; nx: 0.0; ny: 0.0; nz: -1.0; color : $FFF00000 ),
    (x : 1.0; y :-1.0; z :-1.0; nx: 0.0; ny: 0.0; nz: -1.0; color : $FFF00000 ),
    (x :-1.0; y :-1.0; z :-1.0; nx: 0.0; ny: 0.0; nz: -1.0; color : $FFF00000 ),

    (x : 1.0; y :-1.0; z : 1.0; nx: 0.0; ny: 0.0; nz: 1.0; color : $FFF00000 ),  // Hinten
    (x : 1.0; y : 1.0; z : 1.0; nx: 0.0; ny: 0.0; nz: 1.0; color : $FFF00000 ),
    (x :-1.0; y : 1.0; z : 1.0; nx: 0.0; ny: 0.0; nz: 1.0; color : $FFF00000 ),
    (x :-1.0; y : 1.0; z : 1.0; nx: 0.0; ny: 0.0; nz: 1.0; color : $FFF00000 ),
    (x :-1.0; y :-1.0; z : 1.0; nx: 0.0; ny: 0.0; nz: 1.0; color : $FFF00000 ),
    (x : 1.0; y :-1.0; z : 1.0; nx: 0.0; ny: 0.0; nz: 1.0; color : $FFF00000 ),

    (x :-1.0; y : 1.0; z :-1.0; nx: 0.0; ny: 1.0; nz: 0.0; color : $FFF00000 ),  // Oben
    (x :-1.0; y : 1.0; z : 1.0; nx: 0.0; ny: 1.0; nz: 0.0; color : $FFF00000 ),
    (x : 1.0; y : 1.0; z : 1.0; nx: 0.0; ny: 1.0; nz: 0.0; color : $FFF00000 ),
    (x : 1.0; y : 1.0; z : 1.0; nx: 0.0; ny: 1.0; nz: 0.0; color : $FFF00000 ),
    (x : 1.0; y : 1.0; z :-1.0; nx: 0.0; ny: 1.0; nz: 0.0; color : $FFF00000 ),
    (x :-1.0; y : 1.0; z :-1.0; nx: 0.0; ny: 1.0; nz: 0.0; color : $FFF00000 ),

    (x : 1.0; y :-1.0; z :-1.0; nx: 0.0; ny: -1.0; nz: 0.0; color : $FFF00000 ),  // Unten
    (x : 1.0; y :-1.0; z : 1.0; nx: 0.0; ny: -1.0; nz: 0.0; color : $FFF00000 ),
    (x :-1.0; y :-1.0; z : 1.0; nx: 0.0; ny: -1.0; nz: 0.0; color : $FFF00000 ),
    (x :-1.0; y :-1.0; z : 1.0; nx: 0.0; ny: -1.0; nz: 0.0; color : $FFF00000 ),
    (x :-1.0; y :-1.0; z :-1.0; nx: 0.0; ny: -1.0; nz: 0.0; color : $FFF00000 ),
    (x : 1.0; y :-1.0; z :-1.0; nx: 0.0; ny: -1.0; nz: 0.0; color : $FFF00000 ),

    (x :-1.0; y :-1.0; z : 1.0; nx: -1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),  // Links
    (x :-1.0; y : 1.0; z : 1.0; nx: -1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),
    (x :-1.0; y : 1.0; z :-1.0; nx: -1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),
    (x :-1.0; y : 1.0; z :-1.0; nx: -1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),
    (x :-1.0; y :-1.0; z :-1.0; nx: -1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),
    (x :-1.0; y :-1.0; z : 1.0; nx: -1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),

    (x : 1.0; y :-1.0; z :-1.0; nx: 1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),  // Rechts
    (x : 1.0; y : 1.0; z :-1.0; nx: 1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),
    (x : 1.0; y : 1.0; z : 1.0; nx: 1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),
    (x : 1.0; y : 1.0; z : 1.0; nx: 1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),
    (x : 1.0; y :-1.0; z : 1.0; nx: 1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ),
    (x : 1.0; y :-1.0; z :-1.0; nx: 1.0; ny: 0.0; nz: 0.0; color : $FFF00000 ));

{ ---------------------------------------------------------------------------
                        Öffentliche globale Variablen
  --------------------------------------------------------------------------- }

var
  Direct3DForm: TDirect3DForm;  // Die Instanz von TDirect3DForm ist öffentlich für alle
                                // anderen Units erreichbar

{ ---------------------------------------------------------------------------
                        IMPLEMENTATIONSTEIL DER UNIT
  --------------------------------------------------------------------------- }

implementation
```

```
{ ---------------------------------------------------------------------------
                              Compiler-Schalter
  --------------------------------------------------------------------------- }

{$R *.dfm}

{ ---------------------------------------------------------------------------
                    Liste aller private eingebundenen Units
  --------------------------------------------------------------------------- }

uses MDIMainSource;


{ ---------------------------------------------------------------------------

  Methodenname:         FormClose
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var TCloseAction
        Name:           Action
        Bedeutung:      Gibt an, wie das Fenster geschlossen werden ssoll

  Methodenbeschreibung:
  Methode wird beim Beenden des Formulars aufgerufen.

  --------------------------------------------------------------------------- }

procedure TDirect3DForm.FormClose(Sender: TObject;
  var Action: TCloseAction);

begin
  D3DKillScene;         // Abschalten der Szene
  D3DShutdown;          // Beenden von Direct3D
  D3DActive := false;   // Signalisieren, dass es keine Instanz von dem Fenster gibt
  Action := caFree;     // Speicher freigeben
end;


{ ---------------------------------------------------------------------------

  Methodenname:         FormCreate
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird beim erzeugen des Formulars aufgerufen und Initialisiert
  Variablen. Zusätzlich registriert er den IdleHandler

  --------------------------------------------------------------------------- }

procedure TDirect3DForm.FormCreate(Sender: TObject);

begin
  lpd3d := nil;
  lpd3ddevice := nil;
  MyVB := nil;
  Application.OnIdle := MyIdleHandler;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         FatalError
  Parameter
        Typ:            HResult
        Name:           hr
        Bedeutung:      Wert des Handle-Result der Direct3D Szene
        Typ:            string
        Name:           FehlerMsg
        Bedeutung:      String der Fehlermeldung

  Methodenbeschreibung:
  Wird aufgerufen um eine Fehlermeldung an den Benutzer auszugeben.
```

```
  ------------------------------------------------------------------------- }

procedure TDirect3DForm.FatalError(hr : HResult; FehlerMsg : string);

var s : string;

begin

  // Hole die Fehlermeldung
  if hr <> 0 then
   s := D3DXErrorString(hr) + #13 + FehlerMsg
  else
   s := FehlerMsg;

  D3DKillScene;                         // Szene abschalten
  D3DShutdown;                          // Direct3D beenden
  MessageDlg(s, mtError, [mbOK], 0);    // Fehlermeldung ausgeben
  Close;                                // Formular beenden
end;


{ -------------------------------------------------------------------------

  Methodenname:         D3DInit

  Methodenbeschreibung:
  Diese Methode initialisiert Direct3D und die damit verbundene Kommunikation
  mit den COM-Objekten.

  ------------------------------------------------------------------------- }

procedure TDirect3DForm.D3DInit;

var hr    : HRESULT;                  // Speichert die Ergebnisse der Verarbeitung
    d3dpp : TD3DPRESENTPARAMETERS;    // Struktur zum festlegen von Einstellungen
    d3ddm : TD3DDISPLAYMODE;          // Struktur zur Anzeige

begin

  // Erzeuge eine Verbindung zu Direct3D
  lpd3d := Direct3DCreate8(D3D_SDK_VERSION);

  // Falls Erzeugung fehlschläge, gib eine Fehlermeldung aus
  if(lpd3d = nil) then
   FatalError(0,'Fehler beim Erstellen von Direct3D!');

  // Datenstrukturen mit null überschreiben, verhindert Seiteneffekte
  ZeroMemory(@d3dpp, sizeof(d3dpp));
  ZeroMemory(@light, sizeof(light));
  ZeroMemory(@material, sizeof(material));

  // Definiere Einstellungen für Direct3D
  with d3dpp do
   begin
    SwapEffect := D3DSWAPEFFECT_DISCARD;        // Löschen alter Frames
    hDeviceWindow := Handle;                    // Handle vom Formular übergeben
    BackBufferCount := 1;                       // Ein Backpuffer

    EnableAutoDepthStencil := TRUE;             // Z-Buffer Aktivierung
    AutoDepthStencilFormat := D3DFMT_D16;

    Windowed := TRUE;                           // Im Fenstermodus anzeigen

    hr:=lpd3d.GetAdapterDisplayMode(D3DADAPTER_DEFAULT, d3ddm);

    // Falls Erzeugung fehlschläge, gib eine Fehlermeldung aus
    if failed(hr) then
     FatalError(hr,'Fehler beim Ermitteln des Dislaymodes');

    BackBufferFormat := d3ddm.Format;
   end;

  // Erzeuge Direct3D Device
  hr:=lpd3d.CreateDevice(D3DADAPTER_DEFAULT,
                         D3DDEVTYPE_HAL,
                         Handle,                                // Handle vom Formular
                                                                // übergeben
                         D3DCREATE_SOFTWARE_VERTEXPROCESSING,   // Vertexverarbeitung
```

```
                                                                  // von der Software
                                                                  // Hardwarebeschleunigung
                                                                  // deaktiviert
                        d3dpp,
                        lpd3ddevice);

  // Falls Erzeugung fehlschläge, gib eine Fehlermeldung aus
  if FAILED(hr) then
   FatalError(hr,'Fehler beim Erzeugen des 3D-Device');

end;


{ ---------------------------------------------------------------------------

  Methodenname:         D3DShutdown

  Methodenbeschreibung:
  Wird zum beenden von Direct3D aufgerufen.

  --------------------------------------------------------------------------- }

procedure TDirect3DForm.D3DShutdown;

begin
  if assigned(lpd3ddevice) then lpd3ddevice:=nil;       // Verbindung lösen
  if assigned(lpd3d) then lpd3d:=nil;                   // Verbindung lösen
end;


{ ---------------------------------------------------------------------------

  Methodenname:         D3DInitScene

  Methodenbeschreibung:
  Aufgabe dieser Methode ist das initialisieren der 3D-Szene und deren Objekte.

  --------------------------------------------------------------------------- }

procedure TDirect3DForm.D3DInitScene;

var hr         : HRESULT;       // Speichert die Ergebnisse der Verarbeitung
    vbVertices : pByte;
    ViewMatrix : TD3DXMATRIX;   // Anzeigematrix
    matProj    : TD3DXMATRIX;   // Matrixprojektion

begin

  // Initialisiere nur, wenn ein Direct3D Device erzeugt wurde
  if assigned(lpd3ddevice) then
   with lpd3ddevice do
    begin

    // Setze die Eigenschaften des Materials
    material.Diffuse.r := 1.0;
    material.Diffuse.g := 1.0;
    material.Diffuse.b := 1.0;
    material.Diffuse.a := 1.0;
    material.Ambient.r := 1.0;
    material.Ambient.g := 1.0;
    material.Ambient.b := 1.0;
    material.Ambient.a := 1.0;
    material.Specular.r := 1.0;
    material.Specular.g := 1.0;
    material.Specular.b := 1.0;
    material.Specular.a := 1.0;
    material.Emissive.r := 0.0;
    material.Emissive.g := 0.0;
    material.Emissive.b := 0.0;
    material.Emissive.a := 0.0;
    material.Power := 5.0;
    SetMaterial(material);      // Setze das Material

    // Lichtquelle definieren
    light._Type := D3DLIGHT_POINT;      // Lichtquellentyp
    light.Diffuse.r := 1.0;             // Farbe der Lichtes
    light.Diffuse.g := 1.0;
    light.Diffuse.b := 1.0;
```

```
    light.Range := 1000.0;              // Reichweite
    light.Attenuation0 := 1.0;          // Abnahme des Lichtes

    // Position der Lichtquelle
    light.Position := D3DXVECTOR3(((-1.0 * info.CamX) * info.LightX),
                                  ((1.0 * info.CamY) * info.LightY),
                                  ((-1.0 * info.CamZ) * info.LightZ));

    SetLight(0, light);                 // Setze die Lichtquelle auf Index 0
    LightEnable(0, TRUE);               // Aktiviere das Licht auf Index 0
    SetRenderState(D3DRS_LIGHTING, 1);  // Aktiviere global das Licht

    // Erzeuge Vertex-Buffer für alle Vertexe
    hr := CreateVertexBuffer (sizeof(TMyVertices), // Größe der Vertex-Liste
                              D3DUSAGE_WRITEONLY,  // Nur Schreibzugriffe
                              D3D8T_CUSTOMVERTEX,  // Eigene Vertexe
                              D3DPOOL_MANAGED,
                              MyVB);               // Zeiger zum Buffer

    // Falls Erzeugung fehlschläge, gib eine Fehlermeldung aus
    if FAILED(hr) then
     FatalError(0,'Fehler beim Erstellen des Vertex Buffers');

    with MyVB do
     begin
      hr:=Lock(0,               // Offset des Anfangs
               0,               // Größe des locks( 0 = alles )
               vbVertices,      // Wenn erfolgreich, dann hier ablegen
               0);              // sonstige Flags

      // Falls Erzeugung fehlschläge, gib eine Fehlermeldung aus
      if FAILED(hr) then
       FatalError(0,'Fehler beim Locken des Vertex-Buffers');

      // Kopiere Vertex-Buffer
      Move(MyVertices, vbVertices^, SizeOf(TMyVertices));
      Unlock;
     end;

    // Einstellungen für die Rückseiten der Dreiecke
    SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);
    SetRenderState(D3DRS_ZENABLE, 0);                   // Z-Buffer aktivieren

    // Definieren der Ausrichtung der Kamera
    D3DXMatrixLookAtLH(ViewMatrix,D3DXVECTOR3((-1.0*info.CamX),
                                              (1.0*info.CamY),
                                              (-1.0*info.CamZ)),
                                  D3DXVECTOR3(0.0,0.0,0.0),
                                  D3DXVECTOR3(0.0,1.0,0.0));

    // Transformation für Kamera anwenden
    SetTransform(D3DTS_VIEW,ViewMatrix);

    D3DXMatrixPerspectiveFovLH(matProj,   // Resultierende Matrix
                               D3DX_PI/4, // Sichtwinkel
                               640/480,   // Seitenverhältnis
                               1.0,       // Mindeste Nähe
                               100.0);    // Maximal sichtbare Entfernung
    SetTransform(D3DTS_PROJECTION,matProj );
    end;

end;


{ ---------------------------------------------------------------------------

  Methodenname:         D3DKillScene

  Methodenbeschreibung:
  Wird zum abschalten der Direct3D Szene ausgeführt.

  --------------------------------------------------------------------------- }

procedure TDirect3DForm.D3DKillScene;

begin
  MyVB:=nil;    // Speicher freigeben
end;
```

```
{ ------------------------------------------------------------------------

  Methodenname:         D3DRender

  Methodenbeschreibung:
  Innerhalb dieser Methode wird die Szene und alle daraufhin bezogene
  Verarbeitung beschrieben.

  ------------------------------------------------------------------------ }

procedure TDirect3DForm.D3DRender;

var matWorld     : TD3DXMATRIX;
    rot_matrix   : TD3DXMATRIX;         // Rotationsmatrix
    trans_matrix : TD3DXMATRIX;         // Translationsmatrix
    ViewMatrix   : TD3DXMATRIX;         // Matrix für die Kamera

begin

  // Nur Ausführen, wenn ein Direct3D Device existiert
  if assigned(lpd3ddevice) then
   with lpd3ddevice do
    begin

    // Löschaktion
    Clear(0,                            // Wie viele Rechtecke löschen
          nil,                          // Ganzer Bildschirm
          D3DCLEAR_TARGET,
          D3DCOLOR_XRGB(0,0,0),         // Hintergrund schwarz
          1,                            // Lösche Z-Buffer
          0 );

    // Szenenbeschreibung
    if SUCCEEDED(BeginScene) then
     begin
      SetVertexShader(D3D8T_CUSTOMVERTEX);
      SetStreamSource(0, MyVB, sizeof(TMyVertex));

      // Definition der Matrizen
      D3DXMatrixRotationY(rot_matrix, 0.0);
      D3DXMatrixTranslation(trans_matrix, 0.0, 0.0, 0.0);
      D3DXMatrixMultiply(matWorld, rot_matrix, trans_matrix);

      // Anwendung der Matrix
      SetTransform(D3DTS_WORLD, matWorld);

      DrawPrimitive(D3DPT_TRIANGLELIST, 0, 12); // Zeiche Kubus

      // Definiere die Matrix für die Kameraausrichtung
      D3DXMatrixLookAtLH(ViewMatrix, D3DXVECTOR3((-1.0*info.CamX),
                                                 (1.0*info.CamY),
                                                 (-1.0*info.CamZ)),
                                     D3DXVECTOR3(0.0,0.0,0.0),
                                     D3DXVECTOR3(0.0,1.0,0.0));

      // Wende die Matrix für die Kameraausrichtung an
      SetTransform(D3DTS_VIEW,ViewMatrix);

      // Definiere die Position des Lichtes
      light.Position := D3DXVECTOR3(((-1.0 * info.CamX) * info.LightX),
                                    ((1.0 * info.CamY) * info.LightY),
                                    ((-1.0 * info.CamZ) * info.LightZ));

      // Aktivieren des Lichtes
      SetLight(0, light);
      LightEnable(0, TRUE);
      EndScene;
     end;

    Present(nil, nil, 0, nil);          // Zeige Resultate auf dem Bildschirm
    end;

end;


{ ------------------------------------------------------------------------
```

```
  Methodenname:         MyIdleHandler
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var boolean
        Name:           Done
        Bedeutung:      Gibt an, ob die Verarbeitung fertig ist

  Methodenbeschreibung:
  Dieser Handle wird aufgerufen, wenn keine anderen Daten verarbeitet werden
  müssen.

  ------------------------------------------------------------------------- }

procedure TDirect3DForm.MyIdleHandler (Sender: TObject; var Done: Boolean);

begin
  D3DRender;    // Szene rendern
  Done:=false;  // Weiterhin ausführen
end;


{ -----------------------------------------------------------------------------

  Methodenname:         FormShow
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird ausgeführt, wenn das Fenster sichtbar wird.

  ------------------------------------------------------------------------- }

procedure TDirect3DForm.FormShow(Sender: TObject);

begin
  D3DInit;      // Initialisieren von D3D
  D3DInitScene; // Initialisieren der Szene
  D3DRender;    // Szene darstellen
end;


{ -----------------------------------------------------------------------------

  Methodenname:         FormKeyDown
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var word
        Name:           Key
        Bedeutung:      Enthält den Code für das Zeichen
        Typ:            TShiftState
        Name:           Shift
        Bedeutung:      Gibt an, ob die Shift-Taste gedrückt wurde

  Methodenbeschreibung:
  Wird für das Formular aufgerufen, wenn eine Taste gedrückt wurde.

  ------------------------------------------------------------------------- }

procedure TDirect3DForm.FormKeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);

begin
  if Key=VK_ESCAPE then close;  // Wurde ESCAPE gedrückt, dann Schließe das Fenster
end;

end.
```

Dateiname: MDIChildSource.pas

```
{ ---------------------------------------------------------------------------

  Autor:                Jan Derer
  Datum:                05. 06. 04
  Kontakt:              Darkhabits@aol.com
  Programmname:         VisualL
  Version:              1.0

  Klassenname:          TMDIChildMain
  Version:              1.0
  Kurzbeschreibung:
  Diese Klasse repräsentiert das MDI-Kindfenster.

  --------------------------------------------------------------------------- }

unit MDIChildSource;

{ ---------------------------------------------------------------------------
                BESCHREIBUNG DER SCHNITTSTELLE DER UNIT
  --------------------------------------------------------------------------- }

interface

{ ---------------------------------------------------------------------------
                Liste alle öffentlich eingebundenen Units
  --------------------------------------------------------------------------- }

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, StdCtrls, Buttons, ExtCtrls;

{ ---------------------------------------------------------------------------
                    Deklaration eigener Datentypen
  --------------------------------------------------------------------------- }

type

{ ---------------------------------------------------------------------------
                  Klassenbeschreibung für TMDIChildMain
  --------------------------------------------------------------------------- }

  TMDIChildMain = class(TForm)

    // Auflistung aller eingebundenen Komponenten der Klasse
    PageControl1: TPageControl;
    GrammarSheet: TTabSheet;
    OptionsSheet: TTabSheet;
    Panel2: TPanel;
    RunBtn: TButton;
    Panel3: TPanel;
    GroupBox1: TGroupBox;
    RecursEdit: TEdit;
    BasisAngleEdit: TEdit;
    BasisThickEdit: TEdit;
    AxiomEdit: TEdit;
    Splitter1: TSplitter;
    Panel4: TPanel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    GroupBox3: TGroupBox;
    AACheck: TCheckBox;
    TargaCheck: TCheckBox;
    WidthEdit: TEdit;
    Label12: TLabel;
    HeightEdit: TEdit;
    Label13: TLabel;
    LightCamSheet: TTabSheet;
    GroupBox4: TGroupBox;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    GroupBox5: TGroupBox;
    Label17: TLabel;
```

```
    Label18: TLabel;
    Label19: TLabel;
    Image1: TImage;
    SaveBtn: TButton;
    SaveDialog: TSaveDialog;
    SaveAsBtn: TButton;
    CamXTrack: TTrackBar;
    CamYTrack: TTrackBar;
    CamZTrack: TTrackBar;
    CamZLabel: TLabel;
    CamYLabel: TLabel;
    CamXLabel: TLabel;
    LightXTrack: TTrackBar;
    LightYTrack: TTrackBar;
    LightZTrack: TTrackBar;
    LightZLabel: TLabel;
    LightYLabel: TLabel;
    LightXLabel: TLabel;
    GroupBox2: TGroupBox;
    Splitter2: TSplitter;
    Panel1: TPanel;
    Splitter3: TSplitter;
    Panel5: TPanel;
    CheckBtn: TButton;
    Panel9: TPanel;
    MessageList: TListBox;
    Panel6: TPanel;
    Splitter5: TSplitter;
    Panel7: TPanel;
    EditBtn: TButton;
    DelBtn: TButton;
    Button1: TButton;
    Panel8: TPanel;
    Panel10: TPanel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    RightContextEdit: TEdit;
    LeftContextEdit: TEdit;
    ProHeadEdit: TEdit;
    ConditionEdit: TEdit;
    ProBodyEdit: TEdit;
    AddBtn: TButton;
    Panel11: TPanel;
    ProductionList: TListBox;
    DelAllMsgBtn: TButton;
    Direct3DBtn: TButton;
    Panel12: TPanel;
    Panel13: TPanel;
    Panel14: TPanel;
    Panel15: TPanel;
    Splitter4: TSplitter;
    Add2Btn: TButton;
    UpdateBtn: TButton;
    CloseBtn: TButton;
    CondBtn: TButton;
    Label20: TLabel;
    Button2: TButton;

    // Auflistung aller Methoden für die Ereignisverarbeitung
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Button1Click(Sender: TObject);
    procedure DelBtnClick(Sender: TObject);
    procedure AddBtnClick(Sender: TObject);
    procedure EditBtnClick(Sender: TObject);
    procedure RecursEditChange(Sender: TObject);
    procedure RecursEditKeyPress(Sender: TObject; var Key: Char);
    procedure CheckBtnClick(Sender: TObject);
    procedure SaveBtnClick(Sender: TObject);
    procedure SaveAsBtnClick(Sender: TObject);
    procedure RunBtnClick(Sender: TObject);
    procedure DelAllMsgBtnClick(Sender: TObject);
    procedure Direct3DBtnClick(Sender: TObject);
    procedure CamXTrackChange(Sender: TObject);
```

```
    procedure CamYTrackChange(Sender: TObject);
    procedure CamZTrackChange(Sender: TObject);
    procedure LightXTrackChange(Sender: TObject);
    procedure LightYTrackChange(Sender: TObject);
    procedure LightZTrackChange(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure UpdateBtnClick(Sender: TObject);
    procedure Add2BtnClick(Sender: TObject);
    procedure ProBodyEditKeyPress(Sender: TObject; var Key: Char);
    procedure CloseBtnClick(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure FormCreate(Sender: TObject);
    procedure CondBtnClick(Sender: TObject);

  // Auflistung aller privaten Methoden als Prototypen
  private
    function SyntaxLineCheck(var production: String; var new_production: String) : String;
    function SystemCheck : boolean;
    function saveToCFG : boolean;
    procedure StrToProduction(const str: string; var lcontext: string; var predessor: string;
                        var rcontext: string; var condition: string; var successor: string);
    function TestingForBrackets(const succ: string) : string;
    procedure TestingProductionHeads(const pred: string; var headlist: string;
                                     var headdoublelist: string);
    procedure TestingForExistingSymbols(const succ: string; const lcon: string;
                const rcon: string; const headList: string; var missHeadList: string);
    function TestingHeadParameterList(const pred: string; const cond: string;
                                                          const succ: string) : string;
  public
    { Public-Deklarationen }
  end;

{ ------------------------------------------------------------------------
                          Öffentliche globale Variablen
  ------------------------------------------------------------------------ }

var
  MDIChildMain: TMDIChildMain;  // Die Instanz von TMDIMain ist öffentlich für alle anderen
                                // Units erreichbar

{ ------------------------------------------------------------------------
                          IMPLEMENTATIONSTEIL DER UNIT
  ------------------------------------------------------------------------ }

implementation

{ ------------------------------------------------------------------------
                               Compiler-Schalter
  ------------------------------------------------------------------------ }

{$R *.dfm}

{ ------------------------------------------------------------------------
                    Liste aller private eingebundenen Units
  ------------------------------------------------------------------------ }

uses MDIMainSource, Direct3DSource, TurtleComSource, CondFormSource, ConsoleFormSource;

{ ------------------------------------------------------------------------
           Globale Variablen die nur in dieser Unit erreichbar sind
  ------------------------------------------------------------------------ }

var updProduction: string;      // Speichert die Produktion die Bearbeitet werden soll


{ ------------------------------------------------------------------------

  Methodenname:         TestingHeadParameterList
  Rückgabewert
        Typ:            string
        Bedeutung:      Gibt einen String mit einer Warnmeldung zurück oder
                        eine leere Zeichenkette
  Parameter
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           pred
        Bedeutung:      Enthält den Produktionskopf einer Produktion
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           cond
```

```
        Bedeutung:      Enthält den Bedingungsteil einer Produktion
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           succ
        Bedeutung:      Enthält den Produktionskörper einer Produktion

  Methodenbeschreibung:
  Die Aufgabe dieser Methode ist, die Parameterliste des Produktionskopfs zu
  prüfen, ob die angegebenen Parameter verwendet werden und ob Parameter
  verwendet werden die nicht aufgelistet sind.

  ------------------------------------------------------------------------- }

function TMDIChildMain.TestingHeadParameterList(const pred: string; const cond: string;
                                                const succ: string) : string;

var parameterlist : string;     // Speichert alle Variablen, die in der Parameterliste
                                // enthalten sind
    notfoundlist  : string;     // Enthält alle Variablen einer Produktion, die nicht
                                // deklariert wurden
    s             : string;     // Zwischenspeicher
    parameter     : string;     // Kopie von der Parameterliste, um gefundene Variablen aus
                                // der Liste zu löschen
    i, j          : integer;    // Laufvariablen
    bracket       : boolean;    // Signalisiert das eine Parameterliste durchlaufen wird

begin

 // Variablen Initialisierung
 result := '';
 bracket := false;
 parameterlist := '';
 notfoundlist := '';

 i := Pos('(', pred);   // Suche nachdem Anfang einer Parameterliste
 if i = 0 then          // Ist keine Parameterliste vorhanden,
  Exit;                 // dann Beende die Verarbeitung

 // Schleife durchläufen die Parameterliste der Produktion und entnimmt die Variablen
 for j := i+1 to length(pred)-1 do

  // Leerzeichen und das Komma werden nicht beachtet
  if not(pred[j] in [' ', ',']) then
   parameterlist := parameterlist+pred[j]+',';

 parameter := parameterlist;                    // Zuweisung der Kopie der Parameterliste

 // Schleife durchläuft den Bedingungsteil, um Variablen zu finden
 for i := 1 to length(cond) do

  // Ist das nächste Zeichen eine Variable und kein mögliches Zeichen für eine Ausdruck?
  if not(cond[i] in ['=', '<', '>', '&', '|', '+', '*', '-', '/', ' ', '.', '0'..'9']) then
   begin
    j := Pos(cond[i], parameterlist);   // Suche nach der Position des Zeichens
                                        // in der Parameterliste

    // Wurde ein Eintrag gefunden, dann lösche die Variable aus der Liste
    if j > 0 then
     begin
      s := Copy(parameterlist, 1, j-1) + Copy(parameterlist, j+2, length(parameterlist));
      parameterlist := s;
     end

    // Ansonsten trage die Variable in die Liste der nicht deklarierten Parameter,
    // sofern das Zeichen nicht schon dort eingetragen ist
    else if Pos(cond[i], parameter) = 0 then
     if Pos(cond[i], notfoundlist) = 0 then
      notfoundlist := notfoundlist+cond[i]+',';
   end;

 // Durchlaufe den Produktionskörper, um in Argumentenlisten nach Variablen zu suchen
 for i := 1 to length(succ) do

  // Wurde der Anfang einer Parameterliste gefunden, dann setze die Variable bracket
  if succ[i] = '(' then
   bracket := true

  // Wurde das Ende einer Parameterliste gefunden, dann deaktiviere die Variable bracket
  else if succ[i] = ')' then
```

```
   bracket := false

  // Wird gerade eine Parameterliste durchlaufen und die Symbole stellen keinen Ausdruck dar
  // und damit eine Variable, dann Prüfe diese
  else if not(succ[i] in ['=', '<', '>', '&', '|', '+', '*', '-', '/', ' ', '.', '0'..'9'])
                and bracket then
   begin
    j := Pos(succ[i], parameterlist);   // Suche nach der Position des Zeichens
                                        // in der Parameterliste

    // Wurde ein Eintrag gefunden, dann lösche die Variable aus der Liste
    if j > 0 then
     begin
      s := Copy(parameterlist, 1, j-1) + Copy(parameterlist, j+2, length(parameterlist));
      parameterlist := s;
     end

    // Ansonsten trage die Variable in die Liste der nicht deklarierten Parameter,
    // sofern das Zeichen nicht schon dort eingetragen ist
    else if Pos(succ[i], parameter) = 0 then
     if Pos(succ[i], notfoundlist) = 0 then
      notfoundlist := notfoundlist+succ[i]+',';
   end;

 // Gebe eine entsprechende Nachricht zurück, wenn die Parameterliste noch
 // Parameter enthält (damit nicht verwendet werden) oder wenn in der Variable
 // notfoundlist Parameter enthalen sind (Parameter die nicht deklariert wurden)
 if (length(parameterlist) > 0) and (length(notfoundlist) > 0) then
  result := 'Folgende Parameter werden nicht in der Produktion benutzt: '+Copy(parameterlist,
            1, length(parameterlist)-1)+'! Folgende Parameter sind nicht im'+
            '+Produktionskopf enthalten: '+Copy(notfoundlist, 1, length(notfoundlist)-1)+'!'
 else if length(parameterlist) > 0 then
  result := 'Folgende Parameter werden nicht in der Produktion benutzt: '+Copy(parameterlist,
            1, length(parameterlist)-1)+'!'
 else if length(notfoundlist) > 0 then
  result := 'Folgende Parameter sind nicht im Produktionskopf enthalten: '+Copy(notfoundlist,
            1, length(notfoundlist)-1)+'!';

end;


{ ---------------------------------------------------------------------------

  Methodenname:         TestingForExistingSymbols
  Parameter
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           succ
        Bedeutung:      Enthält den Produktionskörper
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           lcon
        Bedeutung:      Enthält den linken Kontext einer Produktion
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           rcon
        Bedeutung:      Enthält den rechten Kontext einer Produktion
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           headList
        Bedeutung:      Ist eine Liste in der alle Symbole enthalten sind für die
                        bisher eine Produktion gefunden wurde
        Typ:            var string
        Name:           missHeadList
        Bedeutung:      Zeiger auf eine String-Liste in dem alle Symbole enthalten
                        sind für die keine Produktion gefunden wurde

  Methodenbeschreibung:
  Diese Methode prüft eine Produktion nach deren enthaltenen Symbolen und
  ob es für die Symbole eine Produktion existiert.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.TestingForExistingSymbols(const succ: string; const lcon: string;
               const rcon: string; const headList: string; var missHeadList: string);

var i       : integer;  // Laufvariable
    bracket : boolean;  // Wird gesetzt, wenn im Produktionskörper eine Parameterliste
                        // gefunden wurde

begin
 bracket := false;      // Initialisierung
```

```
 // Wenn es einen linken Kontext gibt, dann durchlaufe diesen und Prüfe ob ein Symbol
 // enthalten ist, für das noch keine Produktion gefunden wurde und speichere es
 // in der Liste missHeadList ab
 if length(lcon) > 0 then
  for i := 1 to length(lcon) do
   if not((succ[i] in ['c', 'F', 'f', 'Z', 'z', 'g', '.', '+', '-', '&', '^', '/', '\',
           '|', '%', '$', '~', 't', '"', #39, ';', ':', '?', '|', '[', ']', '{', '}', ' '])
           or (Pos(succ[i], headList) > 0) or (Pos(succ[i], missHeadList) > 0)) then
    missHeadList := missHeadList+succ[i]+',';

 // Wenn es einen rechten Kontext gibt, dann durchlaufe diesen und Prüfe ob ein Symbol
 // enthalten ist, für das noch keine Produktion gefunden wurde und speichere es
 // in der Liste missHeadList ab
 if length(rcon) > 0 then
  for i := 1 to length(rcon) do
   if not((succ[i] in ['c', 'F', 'f', 'Z', 'z', 'g', '.', '+', '-', '&', '^', '/', '\',
           '|', '%', '$', '~', 't', '"', #39, ';', ':', '?', '|', '[', ']', '{', '}', ' '])
           or (Pos(succ[i], headList) > 0) or (Pos(succ[i], missHeadList) > 0)) then
    missHeadList := missHeadList+succ[i]+',';

 // Durchlaufe den Produktionskörper und suche Symbolen
 for i := 1 to length(succ) do
  if succ[i] = '(' then         // Wurde der Anfang einer Parameterliste gefunden,
   begin                        // dann setze die Variable bracket
    bracket := true;
    continue;
   end
  else if succ[i] = ')' then    // Wurde das Ende einer Parameterliste gefunden,
   begin                        // dann deaktiviere die Variable bracket
    bracket := false;
    continue;
   end
  else if bracket then          // Überspringe den Inhalt einer Parameterliste
   continue

  // Prüfe ob das Zeichen ein Symbol darstellt und speichere es in der Liste
  // missHeadList ab, wenn es keine Produktion dafür existiert
  else if not((succ[i] in ['c', 'F', 'f', 'Z', 'z', 'g', '.', '+', '-', '&', '^', '/',
              '\', '|', '%', '$', '~', 't', '"', #39, ';', ':', '?', '|', '[', ']',
              '{', '}', ' '])
              or (Pos(succ[i], headList) > 0) or (Pos(succ[i], missHeadList) > 0)) then
   missHeadList := missHeadList+succ[i]+',';
end;


{ ---------------------------------------------------------------------------

  Methodenname:         TestingProductionHeads
  Parameter
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           pred
        Bedeutung:      Enthält den Produktionskopf
        Typ:            var string
        Name:           headList
        Bedeutung:      Zeiger auf eine String-Liste mit allen vorhandenen
                        Produktionsköpfen
        Typ:            var string
        Name:           headdoublelist
        Bedeutung:      Zeiger auf eine String-Liste, in der alle Symbole enthalten
                        sind die mehr als eine Produktion aufweisen können

  Methodenbeschreibung:
  Prüft ob für ein Symbol mehr als eine Produktion existert.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.TestingProductionHeads(const pred: string; var headlist: string;
                                               var headdoublelist: string);

var i : integer;        // Zwischenspeicher für Ergebnisse
    s : string;         // Zwischenspeicher für Ergebnisse

begin
 i := Pos('(', pred);   // Suche nachdem Anfang einer Parameterliste im Produktionskopf
 if i > 0 then          // Wurde eine gefunden, dann kopiere das Symbol heraus
  s := Copy(pred, 1, i-1)
 else                   // Wurde keine gefunden, dann übernehme den Produktionskopf
```

```
  s := pred;

 i := Pos(s+',', headlist);     // Suche nach einem Eintrag des Symbols in der Liste headList

 // Wurde ein Eintrag gefunden, dann speichere das Symbol in die headdoublelist Liste,
 // ansonsten trage die Variable in die headList Liste ein
 if i > 0 then
  headdoublelist := headdoublelist + s+','
 else
  headlist := headlist+s+',';
end;


{ ---------------------------------------------------------------------------

  Methodenname:         TestingForBrackets
  Rückgabewert
        Typ:            string
        Bedeutung:      Gibt einen String mit einer Warnmeldung zurück oder
                        eine leere Zeichenkette
  Parameter
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           succ
        Bedeutung:      Enthält den Produktionskörper einer Produktion

  Methodenbeschreibung:
  Es wird geprüft, ob die Anzahl der öffnenden und schließenden, eckigen und
  geschweifen Klammern gleich ist.

  --------------------------------------------------------------------------- }

function TMDIChildMain.TestingForBrackets(const succ: string) : string;

var lbracket : integer;         // Enthält die Anzahl der eckigen öffnenden Klammern
    rbracket : integer;         // Enthält die Anzahl der eckigen schließenden Klammern
    lpolygon : integer;         // Enthält die Anzahl der geschweiften öffnenden Klammern
    rpolygon : integer;         // Enthält die Anzahl der geschweiften schließenden Klammern
    i        : integer;         // Laufvariable

begin

 // Initialisierung von Variablen
 lbracket := 0;
 rbracket := 0;
 lpolygon := 0;
 rpolygon := 0;
 result := '';

 // Durchlaufe den Produktionskörper und erhöhe den Wert einer Variable, wenn
 // eine Klammer entsprechend der Bedeutung der Variable gefunden wurde
 for i := 1 to length(succ) do
  if succ[i] = '[' then
   inc(lbracket)
  else if succ[i] = ']' then
   inc(rbracket)
  else if succ[i] = '{' then
   inc(lpolygon)
  else if succ[i] = '}' then
   inc(rpolygon);

 // Gibt eine entsprechende Nachricht zurück, wenn die Anzahl der öffnenden und
 // schließenden, eckigen und geschweiften Klammern ungleich ist
 if (lbracket <> rbracket) and (lpolygon <> rpolygon) then
  result := 'Die Anzahl der schliessenden und öffnenden, eckigen und geschweiften Klammern+'
            +' ist ungleich!'
 else if lbracket <> rbracket then
  result := 'Die Anzahl der schliessenden und öffnenden eckigen Klammern ist ungleich!'
 else if lpolygon <> rpolygon then
  result := 'Die Anzahl der schliessenden und öffnenden geschweiften Klammern ist ungleich!';

end;


{ ---------------------------------------------------------------------------

  Methodenname:         StrToProduction
  Parameter
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
```

```
        Name:           str
        Bedeutung:      String der zerlegt werden soll
        Typ:            var string
        Name:           lcontext
        Bedeutung:      Enthält am Schluss den linken Kontext
        Typ:            var string
        Name:           predessor
        Bedeutung:      Enthält am Schluss den Produktionskopf
        Typ:            var string
        Name:           rcontext
        Bedeutung:      Enthält am Schluss den rechten Kontext
        Typ:            var string
        Name:           condition
        Bedeutung:      Enthält am Schluss den Bedingungsteil
        Typ:            var string
        Name:           successor
        Bedeutung:      Enthält am Schluss den Produktionskörper

  Methodenbeschreibung:
  Zerlegt einen gegebenen String, in die Einzelteile einer Produktion.

  ------------------------------------------------------------------------- }

procedure TMDIChildMain.StrToProduction(const str: string; var lcontext: string;
                var predessor: string; var rcontext: string; var condition: string;
                var successor: string);

var lcon : integer;     // Enthält die Position, an der der linke Kontext anfängt
    rcon : integer;     // Enthält die Position, an der der recte Kontext anfängt
    cond : integer;     // Enthält die Position, an der der Bedingungsteil anfängt
    prob : integer;     // Enthält die Position, an der der Produktionskörper anfängt

begin

 // Die Variablen werden mit den Werten deren Bedeutung belegt
 lcon := Pos(' < ', str);
 rcon := Pos(' > ', str);
 cond := Pos(' : ', str);
 prob := Pos(' -> ', str);

 // Ist ein linker Kontext vorhanden und nicht Bestandteil eines arithmetischen
 // Ausdruckes (weil es auch "kleiner" bedeuten kann), dann kopieren den String
 // heraus und speichere es in lcontext ab
 if (lcon > 0) and ((lcon < cond) or (cond = 0)) then
  lcontext := Copy(str, 1, lcon-1)
 else           // Ansonsten belege den Teil mit einem Wildcard
  lcontext := '*';

 successor := Copy(str, prob+4, length(str));   // Kopiere den Produktionskörper heraus

 // Die folgenden if-Anweisungen lösen den Produktionskopf heraus, wobei beachtet
 // werden muss (um die Position exakt zu bestimmen) welche Teile einer
 // Produktion enthalten sind
 if (rcon > 0) and ((rcon < cond) or (cond = 0)) then
  if lcon = 0 then
   predessor := Copy(str, 1, rcon-1)
  else
   predessor := Copy(str, lcon+3, rcon-3-lcon)
 else if cond > 0 then
  if (lcon = 0) or (lcon > cond) then
   predessor := Copy(str, 1, cond-1)
  else
   predessor := Copy(str, lcon+3, cond-3-lcon)
 else
  if lcon = 0 then
   predessor := Copy(str, 1, prob-1)
  else
   predessor := Copy(str, lcon+3, prob-3-lcon);

 // Wenn ein rechter Kontext vorhaden ist, dann kopiere diesen heraus
 if rcon > 0 then
  if cond = 0 then
   rcontext := Copy(str, rcon+3, prob-3-rcon)
  else
   rcontext := Copy(str, rcon+3, cond-3-rcon)
 else
  rcontext := '*';      // Ansonsten belege den Teil mit einem Wildcard
```

```
 // Wenn ein Bedingungsteil vorhaden ist, dann kopiere diesen heraus
 if cond > 0 then
  condition := Copy(str, cond+3, prob-3-cond)
 else
  condition := '*';     // Ansonsten belege den Teil mit einem Wildcard
end;


{ ---------------------------------------------------------------------------

  Methodenname:         saveToCFG
  Rückgabewert
        Typ:            boolean
        Bedeutung:      Gibt zurück, ob die Verarbeitung erfolgreich war

  Methodenbeschreibung:
  Speichert die aktuelle Einstellung in der Info-Strukur in die CFG-Datei.

  --------------------------------------------------------------------------- }

function TMDIChildMain.saveToCFG : boolean;

var cfg : TextFile;     // Zeiger auf die CFG-Datei

begin
 result := true;        // Initialisierung des Rückgabewertes

 // Dateiverarbeitung
 AssignFile(cfg, 'LPROCESS.CFG');       // Dateinamen den File-Handle zu zuweisen
 {$i-} Rewrite(cfg); {$i+}              // Öffnet Datei zum schreiben und schaltet
                                        // zwischenzeitig die I/O-Prüfung aus
 if IOResult <> 0 then          // Wurde die Datei erfolglos geöffnet,
  begin                         // dann Beende die Verarbeitung
   result := false;
   Exit;
  end;

 // Schreibe den Inhalt der Info-Struktur in die CFG-Datei
 if Info.Bitmap = true then
  WriteLn(cfg, 'Image=bmp')
 else
  WriteLn(cfg, 'Image=Targa');
 if Info.AA = true then
  WriteLn(cfg, 'AA=on')
 else
  WriteLn(cfg, 'AA=off');
 if Info.Files = true then
  WriteLn(cfg, 'Files=on')
 else
  WriteLn(cfg, 'Files=off');
 WriteLn(cfg, 'Width='+IntToStr(Info.Width));
 WriteLn(cfg, 'Height='+IntToStr(Info.Height));
 WriteLn(cfg, 'POV='+Info.PathPOV);
 WriteLn(cfg, 'QPOV='+Info.PathQPOV);
 WriteLn(cfg, 'CamX='+FloatToStr(Info.CamX));
 WriteLn(cfg, 'CamY='+FloatToStr(Info.CamY));
 WriteLn(cfg, 'CamZ='+FloatToStr(Info.CamZ));
 WriteLn(cfg, 'LightX='+FloatToStr(Info.LightX));
 WriteLn(cfg, 'LightY='+FloatToStr(Info.LightY));
 WriteLn(cfg, 'LightZ='+FloatToStr(Info.LightZ));

 CloseFile(cfg);        // Schließe den Zugriff auf die CFG-Datei
end;


{ ---------------------------------------------------------------------------

  Methodenname:         SyntaxLineCheck
  Rückgabewert
        Typ:            string
        Bedeutung:      Gibt einen String mit einer Fehlermeldung zurück oder
                        eine leere Zeichenkette
  Parameter
        Typ:            var string
        Name:           production
        Bedeutung:      Enthält die ursprüngliche Produktion
        Typ:            var string
        Name:           new_production
```

```
        Bedeutung:      Enthält die ursprüngliche Produktion formatiert

  Methodenbeschreibung:
  Prüft eine Produktion nach Fehlern und formatiert den String der Produktion.

  ------------------------------------------------------------------------- }

function TMDIChildMain.SyntaxLineCheck(var production: String; var new_production: String)
                                       : String;

var substr         : String;    // Zwischenspeicher
    pred           : String;    // Enthält den Produktionskopf
    con            : String;    // Enthält den Bedingungsteil
    index1, index2 : integer;   // Zwischenspeicher

begin

 result := '';          // Initialisierung

 index1 := Pos('<', production);               // Sucht die Position des linken Kontext
 substr := Copy(production, 1, index1-2);      // Kopiert den linken Kontext heraus

 // Ist im linken Kontext ein Wildcard und dahiner befinden sich Zeichen, dann gib eine
 // Fehlermeldung aus
 if (Pos('*', substr) > 0) and (length(substr) > 1) then
  begin
   result := 'Der linke Kontext enthält neben dem *-Zeichen, noch weitere Zeichen!';
   Exit;
  end;

 // Ist im linken Kontext eine runde Klammer, dann gib eine Fehlermeldung aus
 if (Pos('*', substr) = 0) and ((Pos('(', substr) > 0) or (Pos(')', substr) > 0)) then
  begin
   result := 'Im Kontext sind keine Klammern erlaubt!';
   Exit;
  end;

 // Wenn kein Wildcardsymbol enthalten ist, dann formatiere den String
 if Pos('*', substr) = 0 then
  new_production := substr+' < ';

 index2 := Pos('>', production);               // Sucht die Position des rechten Kontext
 pred := Copy(production, index1+2, index2-3-index1);   // Kopiert den Produktionskopf heraus

 // Prüft, ob eine Klammer von der Parameterliste fehlt und gibt eine Fehlermeldung aus
 if (Pos('(', pred) > 0) xor (Pos(')', pred) > 0) then
  begin
   result := 'Es fehlt eine Klammer!';
   Exit;
  end;

 index1 := Pos(':', production);               // Sucht die Position des Bedingungsteils
 substr := Copy(production, index2+2, index1-3-index2); // Kopiert den rechten Kontext heraus

 // Ist im rechten Kontext ein Wildcard und dahiner befinden sich Zeichen, dann gib
 // eine Fehlermeldung aus
 if (Pos('*', substr) > 0) and (length(substr) > 1) then
  begin
   result := 'Der rechte Kontext enthält neben dem *-Zeichen, noch weitere Zeichen!';
   Exit;
  end;

 // Ist im rechten Kontext eine runde Klammer, dann gib eine Fehlermeldung aus
 if (Pos('*', substr) = 0) and ((Pos('(', substr) > 0) or (Pos(')', substr) > 0)) then
  begin
   result := 'Im Kontext sind keine Klammern erlaubt!';
   Exit;
  end;

 // Wenn kein Wildcardsymbol enthalten ist, dann formatiere den String
 if Pos('*', substr) = 0 then
  substr := ' > '+substr
 else
  substr := '';

 index2 := Pos('->', production);              // Sucht die Position des Produktionskörpers
 con := Copy(production, index1+2, index2-3-index1);    // Kopiert den Bedingungsteil heraus
```

```
 // Ist im Bedingungsteil ein Wildcard und dahiner befinden sich Zeichen, dann gib
 // eine Fehlermeldung aus
 if (length(con) > 1) and (Pos('*', con) > 0) then
  begin
   result := 'Die Bedingung enthält neben dem *-Zeichen, noch weitere Zeichen!';
   Exit;
  end;

 // Es wird geprüft, ob eine Parameterliste vorhanden ist, wenn ein Bedingungsteil
 // vorhanden ist
 if (Pos('*', con) = 0) and ((Pos('(', pred) = 0) or (Pos(')', pred) = 0)) then
  begin
   result := 'Für die Bedingung sind keine Parameter vorhanden!';
   Exit;
  end;

 // Wenn ein Wildcardsymbol enthalten ist, dann wird kein Bedingungsteil angefügt
 if Pos('*', con) > 0 then
  con := ' ->'
 else
  begin
   result := MDIMain.testingCondition(con);     // Prüfte ob die Bedingung fehlerfrei ist
   con := ' : '+con+' ->';                      // Formatiere den Bedingungsteil
   if result <> '' then                         // Wenn ein Fehler in der Bedingung
                                                // gefunden wurde,
    exit;                                       // dann Beende die Verarbeitung
  end;
 result := MDIMain.testingProductionHead(pred); // Prüfe den Produktionskopf

 // Wurde keine Fehlermeldung erzeugt, dann Beende die Verarbeitung ohne die
 // endgültige Speicherung des formatierten Strings
 if result <> '' then
  exit;

 // Weist den endgültigen formatierten String zu
 new_production := new_production + pred + substr + con + Copy(production, index2+2,
                       length(production));

end;


{ ---------------------------------------------------------------------------

  Methodenname:         SystemCheck
  Rückgabewert
        Typ:            boolean
        Bedeutung:      Gibt zurück, ob die Verarbeitung erfolgreich war

  Methodenbeschreibung:
  Prüft die Produktionen, ob Schwächen vorhanden sind.

  --------------------------------------------------------------------------- }

function TMDIChildMain.SystemCheck : boolean;

var i, k        : integer;      // Laufvariable
    lcon        : string;       // Enthält den linken Kontext der Produktion
    pred        : string;       // Enthält den Produktionskopf der Produktion
    rcon        : string;       // Enthält den rechten Kontext der Produktion
    cond        : string;       // Enthält den Bedingungsteil der Produktion
    succ        : string;       // Enthält den Produktionskörper der Produktion
    s           : string;       // Zwischenspeicher
    missProd    : string;       // Liste aller Symbole die keine Produktionen haben
    heads       : string;       // Liste aller Symbole die eine Produktion haben
    headdoubles : string;       // Liste aller Symbole die mehrere Produktionen haben

begin

 // Initialisierung der Variablen
 result := true;
 k := 0;
 missProd := '';
 heads := '';
 headdoubles := '';
 MessageList.Clear;

 // Durchlaufe die Liste alle Produktionen
 for i := 0 to ProductionList.Count-1 do
```

```
  begin

   // Zerlege ein Produktionsstring in seine Einzelteile
   StrToProduction(ProductionList.Items[i], lcon, pred, rcon, cond, succ);

   // Teste den Produktionskopf auf Fehler
   TestingProductionHeads(pred, heads, headdoubles);

   // Teste die Produktionsteile nach Symbolen die keine Produktion haben
   TestingForExistingSymbols(succ, lcon, rcon, heads, missProd);

   // Teste die Produktion, nach der Anzahl deren Klammern
   s := TestingForBrackets(succ);
   if s <> '' then      // Ist die Klammernanzahl ungleich?
    begin               // Gebe eine Nachricht aus
     MessageList.Items.Add('Warnung in Regel '+IntToStr(i+1)+' !');
     MessageList.Items.Add(s);
     result := false;
    end;

   // Teste die Parameterliste, ob alle Symbole verwendet werden oder nicht
   // deklarierte verwendet werden
   s := TestingHeadParameterList(pred, cond, succ);
   if s <> '' then      // Ist die Prüfung erfolgreich gewesen,
    begin               // dann gebe eine Nachricht aus
     MessageList.Items.Add('Warnung in Regel '+IntToStr(i+1)+' !');
     MessageList.Items.Add(s);
     result := false;
    end;
  end;

 // Wurde für eine Symbol mehrere Produktionen gefunden?
 if length(headdoubles) > 0 then
  begin                 // Dann gebe eine Nachricht aus
   MessageList.Items.Add('Warnung: Für folgende Symbole gibt es mehrere Produktionen : '+
                                Copy(headdoubles, 1, length(headdoubles)-1));
   result := false;
  end;
 s := '';

 // Sind Symbole vorhanden, für die im ersten Durchgang keine Produktionen gefunden wurde?
 if length(missProd) > 0 then

  // Dann Durchlaufe die Liste und prüfe ob nach der Aufnahme aller Symbole, nun
  // Produktionen für das Symbol vorhanden sind
  for i := 1 to (length(missProd) div 2) do
   begin
    if i = 1 then
     k := 1
    else
     k := k +2;
    if Pos(missProd[k], heads) = 0 then
     s := s+Copy(missProd, k, 2);
   end;
 missProd := s;

 // Sind weiterhin Symbole vorhanden, für die keine Produktion existieren, dann
 // gib eine Nachricht aus
 if length(missProd) > 0 then
  begin
   MessageList.Items.Add('Warnung: Für folgende Symbole existiert keine Produktion : '+
                                Copy(missProd, 1, length(missProd)-1));
   result := false;
  end;

end;


{ ---------------------------------------------------------------------------

  Methodenname:         FormClose
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var TCloseAction
        Name:           Action
        Bedeutung:      Gibt an, wie das Fenster geschlossen werden ssoll
```

```
  Methodenbeschreibung:
  Methode wird beim Beenden des Formulars aufgerufen.

  ------------------------------------------------------------------------- }

procedure TMDIChildMain.FormClose(Sender: TObject;
var Action: TCloseAction);

begin
 if D3DActive then      // Falls ein Direct3D-Fenster offen ist, dann schließe es
  D3DForm.Close;
 if TurtleActive then   // Falls ein Fenster mit den Turtle-Kommandos offen ist,
                        // dann schließe es
  TurtleForm.Close;
 if CondActive then     // Falls ein Fenster mit den Ausdrücken für die Bedingung offen ist,
                        // dann schließe es
  CondForm.Close;
 Action := caFree;      // Gib den Speicher frei
end;


{ -------------------------------------------------------------------------

  Methodenname:         Button1Click
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird ausgeführt, wenn der Button zum Löschen aller Produktionen
  geklickt wird.

  ------------------------------------------------------------------------- }

procedure TMDIChildMain.Button1Click(Sender: TObject);

begin

 // Wenn Produktionen vorhanden sind, dann Frage nach, ob diese wirklich
 // gelöscht werden sollen
 if ProductionList.Items.Count > 0 then
  if MessageDlg('Wollen Sie wirklich alle Produktionen löschen?', mtConfirmation,
                [mbYes, mbNo], 0) = mrYes then
   begin
    ProductionList.Clear;

    // Da das L-System verändert wurde, wird der Button zum speichern freigegeben
    // und das L-System als modifiziert gekennzeichnet
    SaveBtn.Enabled := true;
    if Caption[length(Caption)] <> '*' then
     Caption := Caption+'*';
   end;
end;


{ -------------------------------------------------------------------------

  Methodenname:         DelBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Die Methode wird aufgerufen, wenn der Button zum löschen einer einzelnen
  Produktion geklickt wird.

  ------------------------------------------------------------------------- }

procedure TMDIChildMain.DelBtnClick(Sender: TObject);

begin

 // Ist eine Produktion ausgewählt worden,
 if ProductionList.ItemIndex > -1 then
  begin
```

```
   // dann lösche diese
   ProductionList.Items.Delete(ProductionList.ItemIndex);

   // Da das L-System verändert wurde, wird der Button zum speichern freigegeben
   // und das L-System als modifiziert gekennzeichnet
   SaveBtn.Enabled := true;
   if Caption[length(Caption)] <> '*' then
    Caption := Caption+'*';
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         AddBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Die Methode wird aufgerufen, wenn eine Produktion hinzugefügt werden soll und
  auf den Button "Hinzufügen" geklickt wird.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.AddBtnClick(Sender: TObject);

var production     : String;    // Produktion aus der Eingabemaske
    new_production : String;    // Formatierte Produktion für die Produktionsliste
    msg            : String;    // Nachricht für Fehlermeldungen die ausgegeben werden
    i              : integer;   // Laufvariable

begin

 // Prüfe, ob in allen Editierfelder etwas eingetragen wurde
 if (LeftContextEdit.Text = '') or (RightContextEdit.Text = '') or (ProHeadEdit.Text = '')
       or (ProBodyEdit.Text = '') or (ConditionEdit.Text = '') then
  begin
   MessageDlg('Die Produktion ist nicht vollständig!', mtError, [mbOk], 0);
   Exit;
  end;

 // Baue den String für die Produktion zusammen
 production := trim(LeftContextEdit.Text)+' < '+trim(ProHeadEdit.Text)+' > '+
               trim(RightContextEdit.Text)+' : '+trim(ConditionEdit.Text)+' -> '+
               trim(ProBodyEdit.Text);

 // Prüfe die Produktion
 msg := SyntaxLineCheck(production, new_production);

 // Wurde kein Fehler gefunden, dann Prüfe ob der String schon existiert in
 // der Produktionsliste, ansonsten füge den String ein
 if msg = '' then
  begin
   for i := 0 to ProductionList.Count-1 do      // Durchlaufen der Produktionen
    begin
     if CompareText(new_production, ProductionList.Items[i]) = 0 then
      begin
       MessageDlg('Diese Produktion existiert schon!', mtWarning, [mbOk], 0);
       Exit;
      end;
    end;
   ProductionList.Items.Add(new_production);
  end

 // Wurde ein Fehler gefunden, dann gib diesen aus und Beende die Verarbeitung
 else
  begin
   MessageDlg(msg, mtError, [mbOk], 0);
   Exit;
  end;

 // Markiere, dass das L-System verändert wurde
 if Caption[length(Caption)] <> '*' then
  Caption := Caption+'*';
 LeftContextEdit.Text := '*';
```

```
 RightContextEdit.Text := '*';
 ConditionEdit.Text := '*';
 ProHeadEdit.Text := '';
 ProBodyEdit.Text := '';
 SaveBtn.Enabled := true;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         EditBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Wenn eine Produktion bearbeitet werden soll, dann Rufe diese Funktion auf.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.EditBtnClick(Sender: TObject);

var lcon        : string;       // Enthält den linken Kontext der Produktion
    pred        : string;       // Enthält den Produktionskopf der Produktion
    rcon        : string;       // Enthält den rechten Kontext der Produktion
    cond        : string;       // Enthält den Bedingungsteil der Produktion
    prob        : string;       // Enthält den Produktionskörper der Produktion
    i           : integer;      // Laufvariable

begin

 // Speicher die zu bearbeitende Produktion zwischen, damit UpdateBtnClick
 // auf den String zugreifen kann
 updProduction := ProductionList.Items[ProductionList.ItemIndex];

 // Zerlege die Produktion in seine Einzelteile und zeige diese mit den
 // entsprechenden Editierfelder an
 StrToProduction(ProductionList.Items[ProductionList.ItemIndex], lcon, pred, rcon,
                       cond, prob);
 LeftContextEdit.Text := lcon;
 ProHeadEdit.Text := pred;
 RightContextEdit.Text := rcon;
 ConditionEdit.Text := cond;
 ProBodyEdit.Text := prob;
 UpdateBtn.Visible := true;

 Add2Btn.Visible := true;       // Aktiviere die alternative Hinzufüge-Funktion
 AddBtn.Visible := false;       // Deaktiviere die Standard Hinzufüge-Funktion

 // Passe die Größe des Editierfeldes für den Produktionskörper
 // an die Größe des Strings an
 i := Canvas.TextWidth(pred);
 if i > ProBodyEdit.Width then
  begin
   ProBodyEdit.Width := i+8;
   UpdateBtn.Left := ProBodyEdit.Width + 311;
   AddBtn.Left := UpdateBtn.Left;
   Add2Btn.Left := UpdateBtn.Left + 88;
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         RecursEditChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Diese Methode wird von allen Editierfeldern verwendet und wird ausgeführt
  wenn der Inhalt eines Editierfeldes sich ändert.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.RecursEditChange(Sender: TObject);
```

```
begin

 // Wenn das aufrufende Objekt ein TEdit-Objekt ist, dann markiere das Fenster
 // das sich der Inhalt verändert hat
 if (Sender As TEdit).Modified = true then
  begin
   SaveBtn.Enabled := true;
   if Caption[length(Caption)] <> '*' then
    Caption := Caption+'*';
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         RecursEditKeyPress
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var char
        Name:           Key
        Bedeutung:      Zeiger auf das Zeichen, das eingegeben wurde

  Methodenbeschreibung:
  Bei jeder Zeicheneingabe für Editierfelder die nur Zahlen akzeptieren,
  wird diese Methode aufgerufen.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.RecursEditKeyPress(Sender: TObject; var Key: Char);

begin

 // Lasse nur Zahlen und den Rücklauf als Eingabe zu
 if (Key in ['0'..'9']) or (Key = #8) then
  Key := Key
 else
  Key := #0;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         CheckBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Führt die Prüfung der gesamten Produktionen aus, wenn auf den Button
  "Prüfen" geklickt wird.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.CheckBtnClick(Sender: TObject);

begin
 SystemCheck;   // Führe die Prüfung aus
end;


{ ---------------------------------------------------------------------------

  Methodenname:         SaveBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Speichert den Inhalt des Kindfensters in die LS-Datei ab.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.SaveBtnClick(Sender: TObject);
```

```
var filename    : string;       // Enthält den Dateinamen der LS-Datei
    s           : string;       // Zwischenspeicher
    old_file    : TextFile;     // File-Handle auf die LS-Datei
    new_file    : TextFile;     // File-Handle auf die LS-Datei mit den neuen Inhalt
    can_open    : boolean;      // Speichert, ob old_file geöffnet werden konnte
    abort       : boolean;      // Gibt an ob das Schreiben in die Datei abgebrochen
                                // werden soll
    i           : integer;      // Laufvariable

begin

 // Prüfe, ob der Inhalt verändert wurde
 if Caption[length(Caption)] <> '*' then
  Exit;

 // Führe ein Test des L-Systems aus
 if SystemCheck = false then

  // Falls eine Warnung vorliegt, dann Frage den Anwender ob er wirklich speicher möchte
  if MessageDlg('Es liegen Hinweise zum Lindenmayer-System vor,'+#13+'wollen Sie trotzdem'
               +' fortfahren?', mtInformation, [mbYes, mbNo], 0) = mrNo then
   Exit;

 // Falls es eine neue Datei ist, dann wird eine Dialogbox geöffnet, um den
 // Namen für das zu speicherne L-System einzugeben
 if Pos('Unbenannt', Caption) > 0 then
  begin
    if SaveDialog.Execute = false then
     Exit;
    filename := SaveDialog.FileName;
  end

 // Ansonsten entnehme, den Namen aus der Überschrift des Kindfensters
 else
  filename := Copy(Caption, 1, length(Caption)-1);

 // Öffnen der alten Datei
 AssignFile(old_file, filename);        // Namen den File-Handle zuweisen
 {$i-} Reset(old_file); {$i+}           // Öffnen der Datei zum lesen und zwischenzeitige
                                        // Deaktivierung der I/O-Prüfung
 if IOResult <> 0 then                  // War das öffnen erfolglos,
  can_open := false                     // dann setze nicht die Variable can_open
 else                                   // Ansonsten setze die Variable
  can_open := true;

 // Öffnen der neuen Datei
 AssignFile(new_file, 'temp_VisualL.ls');       // Namen den File-Handle zuweisen
 {$i-} Rewrite(new_file); {$i+}         // Öffnen der Datei zum schreiben und zwischenzeitige
                                        // Deaktivierung der I/O-Prüfung
 if IOResult <> 0 then                  // War das öffnen erfolglos,
  begin                                 // dann gebe eine Meldung aus und Beende
                                        // die Verarbeitung
   MessageDlg('Datei kann nicht gespeichert werden!', mtError, [mbOk], 0);
   Exit;
  end;

 // Konnte die alte Datei geöffnet werden?
 if can_open then
  begin
   abort := false;

   // Durchlauf die alte Datei bis zum ersten Ausdruck des L-Systems und
   // übernehme alle define-Anweisungen und Kommentare in die neue Datei
   while (not eof(old_file)) or (abort  = false) do
    begin
     ReadLn(old_file, s);
     s := trim(s);
     if (Pos('#', s) = 1) or (Pos('/*', s) = 1) then
      WriteLn(new_file, s)
     else if length(s) = 0 then
      continue
     else
      abort := true;
    end;
  end;

 // Schreibe in die neue Datei die Daten für das L-System
```

```
 WriteLn(new_file, RecursEdit.Text);
 WriteLn(new_file, BasisAngleEdit.Text);
 WriteLn(new_file, BasisThickEdit.Text);
 WriteLn(new_file, AxiomEdit.Text);
 WriteLn(new_file);
 for i := 0 to ProductionList.Items.Count-1 do
  WriteLn(new_file, ProductionList.Items[i]);

 CloseFile(new_file);           // Schließe die neue Datei

 // Konnte die alte Datei geöffnet werden, dann schließe und lösche diese
 // und benenne die neue um
 if can_open then
  begin
   CloseFile(old_file);
   DeleteFile(filename);
   RenameFile('temp_VisualL.ls', filename);
  end

 // Benenne die neue Datei nur um in die Zieldatei
 else
  RenameFile('temp_VisualL.ls', filename);

 Caption := filename;           // Aktualisiere die Überschrift des Kindfensters
 SaveBtn.Enabled := false;      // Deaktiviere den Speicher Button

end;


{ ---------------------------------------------------------------------------

  Methodenname:         SaveAsBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Diese Methode wird aufgerufen, wenn auf den Button "Speichern unter"
  geklickt wird.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.SaveAsBtnClick(Sender: TObject);

begin

 // Markiert das L-System das es modifiziert wurde, damit es gespeichert werden kann
 if Pos('Unbenannt', Caption) <> 1 then
  Caption := 'Unbenannt*';
 SaveBtn.Click;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         RunBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Wenn auf den Button "Bild erzeugen" geklickt wird, dann wird diese Methode
  aufgerufen.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.RunBtnClick(Sender: TObject);

begin

 // Führe einen Check des L-Systems durch und wenn Warnungen vorhanden sind
 // Frage den Benutzer nach, ob der ein Bild erzeugen möchten
 if SystemCheck = false then
  if MessageDlg('Es liegen Hinweise zum Lindenmayer-System vor,'+#13+'wollen Sie trotzdem'
               +' fortfahren?', mtInformation, [mbYes, mbNo], 0) = mrNo then
   Exit;
```

```
 // Speicher den Inhalt der Info-Struktur in die CFG-Datei
 if saveToCFG then
  begin

   // Speicher das L-System, wenn es modifiziert wurde
   if Caption[length(Caption)] = '*' then
    SaveBtn.Click;

   // Wenn das L-System nicht gespeichert werden konnte, dann Verlasse die Methode
   if Caption[length(Caption)] = '*' then
    Exit;

   // Initialisiere die JvCreateProcess-Komponente mit dem String für den
   // Aufruf der Fassade und setze das Tag, dass es eine Bilderzeugung ist
   MDIMain.JvCreateProcess1.CommandLine := 'lprocess '+ExtractFilename(Caption);
   MDIMain.JvCreateProcess1.Tag := 1;

   // Baue den Namen für die Bilddatei zusammen
   if Info.Bitmap then
    PicName := Copy(ExtractFilename(Caption), 1, length(ExtractFilename(Caption))-2)+'bmp'
   else
    PicName := Copy(ExtractFilename(Caption), 1, length(ExtractFilename(Caption))-2)+'tga';

   ConsoleFom.ListBox1.Clear;           // Lösche die Konsolenausgabe
   MDIMain.JvCreateProcess1.Run;        // Starte JvCreateProcess
  end

 // Gib eine Meldung aus, wenn die Einstellungen nicht gespeichert werden konnten
 else
  MessageDlg('Einstellungen konnten nicht in die CFG-Datei gespeichert werden!',
               mtError, [mbOk], 0);

end;


{ ---------------------------------------------------------------------------

  Methodenname:         DelAllMsgBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Wird auf den Button zum Löschen aller Nachrichten geklickt, dann wird diese
  Methode aufgerufen.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.DelAllMsgBtnClick(Sender: TObject);

begin

 // Lösche alle Nachrichten, wenn welche vorhanden sind
 if MessageList.Items.Count > 0 then
  MessageList.Clear;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         Direct3DBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Die Methode wird ausgeführt, wenn auf den Button "3D-Ansicht" geklickt wird.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.Direct3DBtnClick(Sender: TObject);

begin

 // Erzeuge ein Instanz des Direct3D-Fensters, wenn noch keine existiert
```

```
 if D3DActive = false then
  begin
   D3DForm := TDirect3DForm.Create(Application);
   D3DActive := true;
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         CamXTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.CamXTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if CamXTrack.Position = 0 then
  begin
   CamXLabel.Caption := '0';
   Info.CamX := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   CamXLabel.Caption := FloatToStrF((CamXTrack.Position * 0.5), ffFixed, 1, 1);
   Info.CamX := CamXTrack.Position * 0.5;
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         CamYTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.CamYTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if CamYTrack.Position = 0 then
  begin
   CamYLabel.Caption := '0';
   Info.CamY := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   CamYLabel.Caption := FloatToStrF((CamYTrack.Position * 0.5), ffFixed, 1, 1);
   Info.CamY := CamYTrack.Position * 0.5;
  end;
end;


{ ---------------------------------------------------------------------------
```

```
  Methodenname:         CamZTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  ------------------------------------------------------------------------- }

procedure TMDIChildMain.CamZTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if CamZTrack.Position = 0 then
  begin
   CamZLabel.Caption := '0';
   Info.CamZ := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   CamZLabel.Caption := FloatToStrF((CamZTrack.Position * 0.5), ffFixed, 1, 1);
   Info.CamZ := CamZTrack.Position * 0.5;
  end;
end;


{ -------------------------------------------------------------------------

  Methodenname:         LightXTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  ------------------------------------------------------------------------- }

procedure TMDIChildMain.LightXTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if LightXTrack.Position = 0 then
  begin
   LightXLabel.Caption := '0';
   Info.LightX := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   LightXLabel.Caption := FloatToStrF((LightXTrack.Position * 0.5), ffFixed, 1, 1);
   Info.LightX := LightXTrack.Position * 0.5;
  end;
end;


{ -------------------------------------------------------------------------

  Methodenname:         LightYTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.
```

```
  ------------------------------------------------------------------------ }

procedure TMDIChildMain.LightYTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if LightYTrack.Position = 0 then
  begin
   LightYLabel.Caption := '0';
   Info.LightY := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   LightYLabel.Caption := FloatToStrF((LightYTrack.Position * 0.5), ffFixed, 1, 1);
   Info.LightY := LightYTrack.Position * 0.5;
  end;
end;


{ ------------------------------------------------------------------------

  Methodenname:         LightZTrackChange
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird aufgerufen, wenn an der TrackBar-Komponente der Wert verändert
  wird. Diese Methode gleicht den Wert mit der Info-Struktur ab.

  ------------------------------------------------------------------------ }

procedure TMDIChildMain.LightZTrackChange(Sender: TObject);

begin

 // Wenn 0 eingestellt wird, dann gebe dieses direkt an
 if LightZTrack.Position = 0 then
  begin
   LightZLabel.Caption := '0';
   Info.LightZ := 0;
  end

 // Ansonsten berechne den neuen Wert
 else
  begin
   LightZLabel.Caption := FloatToStrF((LightZTrack.Position * 0.5), ffFixed, 1, 1);
   Info.LightZ := LightZTrack.Position * 0.5;
  end;
end;


{ ------------------------------------------------------------------------

  Methodenname:         Button2Click
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Die Methode öffnet die Tabelle für die Turtle-Kommandos, wenn auf den Button
  "Turtle-Kommandos" geklickt wird. Vorausgesetzt es gibt nicht schon eine
  Instanz des Fensters.

  ------------------------------------------------------------------------ }

procedure TMDIChildMain.Button2Click(Sender: TObject);

begin
 if TurtleActive = false then
  begin
   TurtleForm := TTurtleComForm.Create(Application);
   TurtleActive := true;
```

```
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         UpdateBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Aktualisiert eine Produktion, wenn auf den Button "Aktualisieren" geklickt
  wird.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.UpdateBtnClick(Sender: TObject);

var i: integer;         // Zwischenspeicher

begin

 // Durchläuft die Produktionsliste nach identischen String
 // Wurde keiner gefunden, dann wird der String in die Liste aufgenommen
 for i := 0 to ProductionList.Count-1 do
  if CompareText(updProduction, ProductionList.Items[i]) = 0 then
   begin
    ProductionList.Items.Delete(i);
    break;
   end;

 // Deaktiviere die Buttons zum Aktualisieren einer Produktion
 Add2Btn.Visible := false;
 AddBtn.Visible := true;
 UpdateBtn.Visible := false;
 AddBtn.Click;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         Add2BtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Fügt eine Produktion hinzu, wenn der Button "Aktualisiere" aktiv ist.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.Add2BtnClick(Sender: TObject);

begin
 Add2Btn.Visible := false;
 AddBtn.Visible := true;
 UpdateBtn.Visible := false;
 AddBtn.Click;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         ProBodyEditKeyPress
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Diese Funktion wird ausgeführt, wenn ein Zeichen im Editierfeld für den
  Produktionskörper eingegeben wird.

  --------------------------------------------------------------------------- }
```

```
procedure TMDIChildMain.ProBodyEditKeyPress(Sender: TObject;
  var Key: Char);

var i : integer;        // Zwischenspeicher

begin

 // Berechne die Breite des Textes des Editierfeldes mit dem neuen Zeichen
 i := Canvas.TextWidth((ProBodyEdit.Text+key));

 // Wurde RETURN gedrückt,
 if (key = #13) and (AddBtn.Visible = true) then
  begin
   AddBtn.Click;        // dann füge die Produktion hinzu
   ProHeadEdit.SetFocus;// Gebe den Eingabefokus an den Editierfeld des Produktionskopf
  end
 else if key = #13 then
  begin
   UpdateBtn.Click;     // dann aktualisiere die Produktion, wenn diese zum
                        // bearbeiten angeklickt wurde
   ProHeadEdit.SetFocus;// Gebe den Eingabefokus an den Editierfeld des Produktionskopf
  end

 // Wird ein Zeichen gelöscht und die berechnete Breite der Zeichenkette ist immer
 // noch größer, als das Editierfeld, dann passe die Größe an
 else if (key = #8) and (i > 169) then
  begin
   ProBodyEdit.Width := i;
   UpdateBtn.Left := ProBodyEdit.Width + 311;
   AddBtn.Left := UpdateBtn.Left;
   Add2Btn.Left := UpdateBtn.Left + 88;
  end

 // Wird ein Zeichen hinzugefügt, dann Passe die Breite des Editierfeldes, an
 // die Breite des Strings
 else if i+9 > ProBodyEdit.Width then
  begin
   ProBodyEdit.Width := i+8;
   UpdateBtn.Left := ProBodyEdit.Width + 311;
   AddBtn.Left := UpdateBtn.Left;
   Add2Btn.Left := UpdateBtn.Left + 88;
  end

 // Ansonsten gebe die Standardwerte
 else
  begin
   ProBodyEdit.Width := 169;
   UpdateBtn.Left := 480;
   AddBtn.Left := 480;
   Add2Btn.Left := 568;
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         CloseBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Schließt das Kindfenster, wenn auf den Button "Schließen" geklickt wird.

  --------------------------------------------------------------------------- }

procedure TMDIChildMain.CloseBtnClick(Sender: TObject);

begin
 Close;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         FormCloseQuery
  Parameter
```

```
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var Boolean
        Name:           CanClose
        Bedeutung:      Gibt an, ob das Formular geschlossen werden soll.

  Methodenbeschreibung:
  Methode wird vor dem Schließen des Formulars aufgerufen.

  ------------------------------------------------------------------------- }

procedure TMDIChildMain.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);

var i : integer;        // Zwischenspeicher

begin
 CanClose := true;

 // Wurde der Inhalt des L-Systems modifiziert, dann wird gefragt, ob die
 // Änderungen gespeichert werden sollen.
 if Pos('*', Caption) > 0 then
  begin
   i := MessageDlg('Das Lindenmayer-System wurde geändert, aber noch nicht gepspeichert.'+
                   #13+'Möchten Sie es jetzt Speichern?', mtInformation,
                   [mbYes, mbNo, mbCancel], 0);
   if i = mrYes then
    SaveBtn.Click
   else if i = mrCancel then
    CanClose := false;
  end;
end;


{ -------------------------------------------------------------------------

  Methodenname:         FormCreate
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird beim erzeugen des Formulars aufgerufen und Initialisiert
  Variablen.

  ------------------------------------------------------------------------- }

procedure TMDIChildMain.FormCreate(Sender: TObject);

begin
 D3DActive := false;
 TurtleActive := false;
end;


{ -------------------------------------------------------------------------

  Methodenname:         CondBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Die Methode öffnet die Tabelle für die Ausdrücke im Bedingungsteil, wenn auf
  den Button "Turtle-Kommandos" geklickt wird. Vorausgesetzt es gibt nicht schon
  eine Instanz des Fensters.

  ------------------------------------------------------------------------- }

procedure TMDIChildMain.CondBtnClick(Sender: TObject);

begin
 if CondActive = false then
  begin
   CondForm := TCondForm.Create(Application);
```

```
   CondActive := true;
  end;
end;

end.
```

Dateiname: MDIMainSource.pas

```
{ ---------------------------------------------------------------------------

  Autor:                Jan Derer
  Datum:                05. 06. 04
  Kontakt:              Darkhabits@aol.com
  Programmname:         VisualL
  Version:              1.0

  Klassenname:          TMDIMain
  Version:              1.0
  Kurzbeschreibung:
  Diese Klasse repräsentiert das MDI-Elternfenster mit einige
  öffentlichen Methode die hilfreiche Funktionen für andere Klassen darstellen.

  --------------------------------------------------------------------------- }

unit MDIMainSource;

{ ---------------------------------------------------------------------------
                BESCHREIBUNG DER SCHNITTSTELLE DER UNIT
  --------------------------------------------------------------------------- }

interface

{ ---------------------------------------------------------------------------
                Liste alle öffentlich eingebundenen Units
  --------------------------------------------------------------------------- }

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ToolWin, ActnMan, ActnCtrls, ActnMenus, ActnList, StdActns,
  ComCtrls, ImgList, Menus, StdCtrls, MDIChildSource, Direct3DSource, TurtleComSource,
  CondFormSource, JvComponent, JvSysComp;

{ ---------------------------------------------------------------------------
                    Deklaration eigener Datentypen
  --------------------------------------------------------------------------- }

type

{ ---------------------------------------------------------------------------
                   Klassenbeschreibung für TMDIMain
  --------------------------------------------------------------------------- }

  TMDIMain = class(TForm)

    // Auflistung aller eingebundenen Komponenten der Klasse
    ActionManager1: TActionManager;
    FileNew: TAction;
    EditCut1: TEditCut;
    EditCopy1: TEditCopy;
    EditPaste1: TEditPaste;
    EditSelectAll1: TEditSelectAll;
    EditUndo1: TEditUndo;
    EditDelete1: TEditDelete;
    HelpAbout: TAction;
    FileOpen1: TFileOpen;
    FileExit1: TFileExit;
    WindowClose1: TWindowClose;
    WindowCascade1: TWindowCascade;
    WindowTileHorizontal1: TWindowTileHorizontal;
    WindowTileVertical1: TWindowTileVertical;
    WindowMinimizeAll1: TWindowMinimizeAll;
    WindowArrange1: TWindowArrange;
    ActionMainMenuBar: TActionMainMenuBar;
    ActionToolBar1: TActionToolBar;
    CoolBar1: TCoolBar;
    StatusBar1: TStatusBar;
    ImageList1: TImageList;
```

```
    OptionsCFG: TAction;
    ImportLS: TAction;
    OpenDialog1: TOpenDialog;
    JvCreateProcess1: TJvCreateProcess;

    // Auflistung aller Methoden für die Ereignisverarbeitung
    procedure HelpAboutExecute(Sender: TObject);
    procedure FileNewExecute(Sender: TObject);
    procedure OptionsCFGExecute(Sender: TObject);
    procedure FileOpen1Accept(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure ImportLSExecute(Sender: TObject);
    procedure JvCreateProcess1Read(Sender: TObject; const S: String);
    procedure JvCreateProcess1Terminate(Sender: TObject;
      ExitCode: Cardinal);

  // Auflistung aller privaten Methoden als Prototypen
  private
    procedure CreateMDIChild(const Name: string);
    function checkLSLine(var line: string) : string;
    procedure defaultCFG;
    procedure fillInfo(var cfg : TextFile);
    procedure setOptionsForChild(var Child : TMDIChildMain);
    function convertRealForTracker(value: real48) : integer;
    procedure openLS(const filename: string);

  // Auflistung aller öffentlichen Methoden als Prototypen
  public
    function testingProductionHead(const ProHead: string) : string;
    function testingCondition(const Condition: string) : string;
  end;

{ ---------------------------------------------------------------------------
                Deklaration der Datenstruktur für die CFG-Datei
  --------------------------------------------------------------------------- }

  TInfo = record
        Bitmap : boolean;       // Gibt an, ob die Grafik als BMP gespeichert werden soll
        AA : boolean;           // Gibt an, ob Anti-Aliasing bei Render verwendet werden soll
        Files : boolean;        // Gibt an, ob alle Dateien aus den Zwischenschritten
                                // gelöscht werden sollen (wird von Visual L nicht verwendet)
        Width : integer;        // Enthält die Breite der Grafik in Pixel
        Height : integer;       // Enthält die Höhe der Grafik in Pixel
        CamX : real48;          // Enthält die Zahl zur relativen Positionierung der Kamera
                                // für die x-Achse
        CamY : real48;          // Enthält die Zahl zur relativen Positionierung der Kamera
                                // für die y-Achse
        CamZ : real48;          // Enthält die Zahl zur relativen Positionierung der Kamera
                                // für die z-Achse
        LightX : real48;        // Enthält die Zahl zur relativen Positionierung des Lichtes
                                // für die x-Achse
        LightY : real48;        // Enthält die Zahl zur relativen Positionierung des Lichtes
                                // für die y-Achse
        LightZ : real48;        // Enthält die Zahl zur relativen Positionierung des Lichtes
                                // für die z-Achse
        PathPOV : string;       // Speichert den Pfad zu POV-Ray als Zeichenkette
        PathQPOV : string;      // Speichert den Pfad zu QuietPOV als Zeichenkette
  end;

{ ---------------------------------------------------------------------------
                          Öffentliche globale Variablen
  --------------------------------------------------------------------------- }

var
  MDIMain: TMDIMain;            // Die Instanz von TMDIMain ist öffentlich für alle anderen
                                // Units erreichbar
  Info: TInfo;                  // Die Datenstruktur ist öffentlich für alle anderen Units
                                // erreichbar
  D3DActive: boolean;           // Gibt an, ob eine Instanz von TDirect3DForm erzeugt wurde
  D3DForm: TDirect3DForm;       // Globale Instanz der Klasse TDirect3DForm
  TurtleActive: boolean;        // Gibt an, ob die Instanz von TTurtleComForm erzeugt wurde
  TurtleForm: TTurtleComForm;   // Globale Instanz der Klasse TTurtleComForm
  CondForm: TCondForm;          // Globale Instanz der Klasse TCondForm
  CondActive: boolean;          // Gibt an, ob die Instanz von TCondForm erzeugt wurde

  PicName: string;              // Globe Variable zum speichern von Zeichenketten (Name der
```

```
                                // Bilddatei)

{ ------------------------------------------------------------------------
                          IMPLEMENTATIONSTEIL DER UNIT
  ------------------------------------------------------------------------ }

implementation

{ ------------------------------------------------------------------------
                               Compiler-Schalter
  ------------------------------------------------------------------------ }

{$R *.dfm}

{ ------------------------------------------------------------------------
                    Liste aller private eingebundenen Units
  ------------------------------------------------------------------------ }

uses AboutSource, CFGSource, ConsoleFormSource, ShowPicSource;


{ ------------------------------------------------------------------------

  Methodenname:         openLS
  Parameter
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           filename
        Bedeutung:      Dateiname der LS-Datei

  Methodenbeschreibung:
  Öffnet eine LS-Datei, prüft den Inhalt und übergibt die Informationen an ein
  MDI-Kindfenster. Wurden Fehler gefunden, werden diese im Nachrichtenfeld des
  Kindfensters erscheinen.

  ------------------------------------------------------------------------ }

procedure TMDIMain.openLS(const filename: string);

var Child       : TMDIChildMain;  // Instanz für ein neues Kindfenster
    f           : TextFile;       // Variable zur Verarbeitung der LS-Datei
    s, t        : string;         // Zwischenspeicherung von Zeichenketten
    i           : integer;        // Laufvariable
    state       : integer;        // Speichert den Zustand der Verarbeitung der Datei
    errorLine   : integer;        // Zählt die Regeln, um eine Fehlerhafte Regel zu benennen
    fl          : single;         // Wird zum testen eines Float-Wertes benötigt

begin

 // Initialisierung von Variablen
 state := 0;
 errorLine := 0;

 // Datei öffnen
 AssignFile(f, filename);       // Zuweisung des Dateinamens an die Variable
 {$i-} Reset(f); {$i+}          // Datei wird versucht zu öffnen um diese Auszulesen
                                // (I/O-Prüfung des Compilers wird dazu deaktiviert)
 if IOResult <> 0 then          // Konnte die Datei nicht geöffnet werden?
  begin                         // Gib eine Fehlermeldung aus und Beende die weitere
                                // Verarbeitung
   MessageDlg('Datei "'+filename+'" konnte nicht geöffnet werden!', mtError, [mbOk], 0);
   Exit;
  end;

 // Erzeuge Kindfenster
 Child := TMDIChildMain.Create(Application);
 Child.Caption := filename;

 // Verarbeitungsschleife für die LS-Datei
 while not eof(f) do
  begin
   readln(f, s);                // Hol eine Zeile aus der Datei
   s := trim(s);                // Schneide Leerzeichen am Anfang und am Ende ab
   if length(s) > 1000 then     // Ist die Zeile zu lang?
    begin                       // Dann überspringe die Zeile und schreib eine Nachricht
                                // ins Kindfenster
     Child.MessageList.Items.Add('Die Zeile ist zu lang!');
     continue;
    end;
```

```
// Enthält die Zeile ein Kommentar, Leerzeile oder eine Präprozessor-Anweisung,
// dann überspringe die Zeile
if (Pos('/*', s) = 1) or (length(s) = 0) or (Pos('#', s) = 1) then
 continue;

i := Pos('/*', s);           // Suche nach ein Kommentarzeichen in einer Zeile mit
                             // einem Ausdruck
if i > 1 then                // Ist ein Kommentar gefunden worden,
 begin
  s := Copy(s, 1, i-1);      // dann Schneide den Kommentar ab und
  s := trim(s);              // entferne die Leerzeichen
 end;

// Die case-Anweisung prüft die Anfangswerte des L-Systems, die am Anfang
// der Datei stehen.
case state of
     0: try                  // Erster gültiger Ausdruck (kein Kommentar oder Leerzeile)
                             // Es wird erwartet, dass der Ausdruck die Zahl für die
                             // Rekursionstiefe repräsentiert
         i := StrToInt(s);   // Der String wird versucht in eine Zahl zu wandeln
                             // Ist es gelungen, wird der Wert an das Kindfenster
                             // weitergegeben und der Zustand erhöht.
         Child.RecursEdit.Text := IntToStr(i);
         Inc(state);
         continue;
        except
         on EConvertError do // Bei einem Konvertierungsfehler, wird eine Nachricht an
          begin              // das Kindfenster übergeben
           Child.MessageList.Items.Add('Keine Rekursionstiefe gefunden!');
           Child.MessageList.Items.Add('Keinen Basiswinkel gefunden!');
           Child.MessageList.Items.Add('Keine Basisstärke gefunden!');
           state := 3;       // Da keine gültige Zahl gefunden wurde, wird als nächstes
                             // nach einem Axiom gesucht
          end;
        end;

     1: try                  // Zweiter gültiger Ausdruck
                             // Es wird erwartet, dass der Ausdruck die Zahl für den
                             // Basiswinkel repräsentiert
         fl := StrToFloat(s);// Der String wird versucht in eine Zahl zu wandeln
                             // Ist es gelungen, wird der Wert an das Kindfenster
                             // weitergegeben und der Zustand erhöht.
         Child.BasisAngleEdit.Text := FloatToStr(fl);
         Inc(state);
         continue;
        except
         on EConvertError do // Bei einem Konvertierungsfehler, wird eine Nachricht an
          begin              // das Kindfenster übergeben
           Child.MessageList.Items.Add('Keinen Basiswinkel gefunden!');
           Child.MessageList.Items.Add('Keine Basisstärke gefunden!');
           state := 3;       // Da keine gültige Zahl gefunden wurde, wird als nächstes
                             // nach einem Axiom gesucht
          end;
        end;

     2: try                  // Dritter gültiger Ausdruck
                             // Es wird erwartet, dass der Ausdruck die Zahl für die
                             // Basisstärke repräsentiert
         i := StrToInt(s);   // Der String wird versucht in eine Zahl zu wandeln
                             // Ist es gelungen, wird der Wert an das Kindfenster
                             // weitergegeben und der Zustand erhöht.
         Child.BasisThickEdit.Text := IntToStr(i);
         Inc(state);
         continue;
        except
         on EConvertError do // Bei einem Konvertierungsfehler, wird eine Nachricht an
          begin              // das Kindfenster übergeben
           Child.MessageList.Items.Add('Keine Basisstärke gefunden!');
           state := 3;       // Da keine gültige Zahl gefunden wurde, wird als nächstes
                             // nach einem Axiom gesucht
          end;
        end;
end;

// An dieser Stelle wird nach einem gültigen Startaxiom gesucht
if state = 3 then
 begin
```

```
     state := 4;        // Erhöhe den Zustand, um danach Produktionen zu verarbeiten

     // Enthält die Zeichenkette Symbole die auf eine Produktion hinweisen?
     if (Pos('<', s) > 0) or (Pos('>', s) > 0) or (Pos(':', s) > 0) or
        (Pos('->', s) > 0) then
      Child.MessageList.Items.Add('Kein Axiom gefunden')
     else
      begin
       Child.AxiomEdit.Text := s;       // Wenn nicht, dann übernehme den Ausdruck
                                        // als Startaxiom
       continue;
      end;
    end;

   // An dieser Stelle werden die Ausdrücke nach einer gültigen Produktion geprüft
   if state = 4 then
    begin
     Inc(errorLine);
     t := checkLSLine(s);               // Prüfe die Zeichenkette, ob es eine gültige
                                        // Produktion ist

     // Wird keine Fehlermeldung zurückgegeben, dann übernehme den String als Produktion.
     if t = '' then
      Child.ProductionList.Items.Add(s)

     // Enthält der String eine Fehlermeldung,dann gib diese aus
     else
      begin
       Child.MessageList.Items.Add('Fehler in Regel '+IntToStr(errorLine)+' !');
       Child.MessageList.Items.Add(t);
      end;
    end;

  end;                          // Ende der Verarbeitungsschleife

 CloseFile(f);                  // Schließe die Datei
 setOptionsForChild(Child);     // Setze die Einstellungen für das Kindfenster aus
                                // der Info-Struktur

end;


{ ---------------------------------------------------------------------------

  Methodenname:         testingCondition
  Rückgabewert
        Typ:            string
        Bedeutung:      Gibt einen String mit einer Fehlermeldung zurück oder
                        eine leere Zeichenkette
  Parameter
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           Condition
        Bedeutung:      Enthält den Bedingungsteil einer Produktion

  Methodenbeschreibung:
  Die Methode prüft, ob ungültige Zeichen im Bedingungsteil enthalten sind.

  --------------------------------------------------------------------------- }

function TMDIMain.testingCondition(const Condition: string) : string;

var i : integer;        // Laufvariable

begin
 result := '';          // Rückgabewert wird mit dem Leerstring initialisiert

 // Durchlauf den kompletten String und Prüfe jedes Zeichen auf seine Gültigkeit
 for i := 1 to length(Condition) do
  if not(Condition[i] in ['a'..'z', 'A'..'Z', '0'..'9', '<'..'>', '-'..'/', ' ',
                          '+', '*']) then
   begin
    result := 'Im Bedingungsteil befindet sich mindestens ein nicht erlaubtes Zeichen!';
    Exit;
   end;

end;
```

```
{ ------------------------------------------------------------------------

  Methodenname:         testingProductionHead
  Rückgabewert
        Typ:            string
        Bedeutung:      Gibt einen String mit einer Fehlermeldung zurück oder
                        eine leere Zeichenkette
  Parameter
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           ProHead
        Bedeutung:      Enthält den Produktionskopf einer Produktion

  Methodenbeschreibung:
  Prüft, ob der Produktionskopf einen gültigen Ausdruck darstellt.

  ------------------------------------------------------------------------ }

function TMDIMain.testingProductionHead(const ProHead: string) : string;

var leftBracket   : integer;    // Stellt die Position der öffnenden runden Klammer da
    rightBracket  : integer;    // Enthält die Position der schließenden runden Klammer
    i             : integer;    // Laufvariable
    ParameterList : string;     // Ist ein Teilstring des Produktionskopf und enthält
                                // die Parameterliste

begin
 result := '';                  // Initialisierung des Rückgabewerts mit dem Leerstring

 leftBracket := Pos('(', ProHead);      // Zuweisung der Position der öffnenden Klammer
 rightBracket := Pos(')', ProHead);     // Zuweisung der Position der schließenden Klammer

 // Wurde nur eine Klammer gefunden?
 if (leftBracket > 0) xor (rightBracket > 0) then
  begin         // Gib Fehlermeldung zurück und Beende die weitere Verarbeitung
   result := 'Im Bedingungsteil befindet sich mindestens ein nicht erlaubtes Zeichen!';
   Exit;
  end

 // Ist die linke Klammer hinter der rechten Klammer?
 else if leftBracket > rightBracket then
  begin         // Gib Fehlermeldung zurück und Beende die weitere Verarbeitung
   result := 'Rechte Klammer steht vor der linken Klammer!';
   Exit;
  end

 // Befindet sich hinter der rechten Klammer noch etwas?
 else if (rightBracket <> length(ProHead)) and (rightBracket > 0) then
  begin         // Gib Fehlermeldung zurück und Beende die weitere Verarbeitung
   result := 'Hinter der rechten Klammer steht noch ein Ausdruck!';
   Exit;
  end

 // Enthält der Produktionskopf eine Parameterliste?
 else if ((leftBracket+1) = rightBracket) then
  begin         // Gib Fehlermeldung zurück und Beende die weitere Verarbeitung
   result := 'In der Klammer befindet sich nichts!';
   Exit;
  end

 // Prüfe die Parameterliste
 else if (leftBracket > 0) and (rightBracket > 0) then
  begin

   // Zuweisung der Parameterliste und abschneiden der Leerzeichen am Anfang und Ende
   ParameterList := trim(Copy(ProHead, leftBracket+1, rightBracket-1-leftBracket));

   // Ist die Parameterliste leer?
   if length(ParameterList) = 0 then
    begin         // Gib Fehlermeldung zurück und Beende die weitere Verarbeitung
     result := 'In der Klammer befindet sich nichts!';
     Exit;
    end;

   // Durchlauf die Parameterliste und prüfe, ob diese nur aus gültigen Zeichen besteht
   for i := 1 to length(ParameterList) do
    if not(ParameterList[i] in['a'..'z', 'A'..'Z', ',', ' ']) then
     begin
      result := 'In der Klammer befindet sich mindestens ein nicht erlaubtes Zeichen!'
```

```
                +'Erlaubt sind nur Buchstaben und das Komma!';
      Exit;
     end;

  end;          // Ende der If-Anweisung

end;


{ ---------------------------------------------------------------------------

  Methodenname:         convertRealForTracker
  Rückgabewert
        Typ:            integer
        Bedeutung:      Gibt eine Ganzzahl für die Tracker-Komponente zurück
  Parameter
        Typ:            real48
        Name:           value
        Bedeutung:      Fließkommazahl aus der CFG-Datei

  Methodenbeschreibung:
  Rechnet eine Fließkommazahl aus der CFG-Datei um, in eine ganze Zahl für die
  Tracker-Komponente. Dabei ist die Auflösung 0.5 und geht von +10 bis -10.

  --------------------------------------------------------------------------- }

function TMDIMain.convertRealForTracker(value: real48) : integer;

begin

 // Wenn die Zahl größer 10 ist, dann weise das Maximum zu und beende
 if value > 10 then
  begin
   result := 20;
   Exit;
  end

 // Wenn die Zahl kleiner -10 ist, dann weise das Minimum zu und beende
 else if value < -10 then
  begin
   result := -20;
   Exit;
  end

 // Wenn die Zahl gleich 0 ist, dann weise die 0 zu und beende
 else if value = 0 then
  begin
   result := 0;
   Exit;
  end;

 // Multipliziere die Fließkommazahl mit zwei und runde den Wert auf, bzw. ab.
 result := round(value * 2);

end;


{ ---------------------------------------------------------------------------

  Methodenname:         setOptionsForChild
  Parameter
        Typ:            var TMDIChildMain
        Name:           Child
        Bedeutung:      Zeiger auf ein erzeugtes Kindfenster

  Methodenbeschreibung:
  Initialisiert das Kindfenster mit den Werten aus der Info-Struktur.

  --------------------------------------------------------------------------- }

procedure TMDIMain.setOptionsForChild(var Child : TMDIChildMain);

begin

 with Child do
  begin
   AACheck.Checked := Info.AA;                  // Anti-Aliasing setzen?
   TargaCheck.Checked := not Info.Bitmap;       // Targa oder Bitmap
```

```
   WidthEdit.Text := IntToStr(Info.Width);      // Breite der Grafik
   HeightEdit.Text := IntToStr(Info.Height);    // HöHe der Grafik

   // Setzen der Werte für die Achsen der Lichtquelle und der Kamera
   CamXTrack.Position := convertRealForTracker(Info.CamX);
   CamXLabel.Caption := FloatToStr(CamXTrack.Position * 0.5);
   CamYTrack.Position := convertRealForTracker(Info.CamY);
   CamYLabel.Caption := FloatToStr(CamYTrack.Position * 0.5);
   CamZTrack.Position := convertRealForTracker(Info.CamZ);
   CamZLabel.Caption := FloatToStr(CamZTrack.Position * 0.5);
   LightXTrack.Position := convertRealForTracker(Info.LightX);
   LightXLabel.Caption := FloatToStr(LightXTrack.Position * 0.5);
   LightYTrack.Position := convertRealForTracker(Info.LightY);
   LightYLabel.Caption := FloatToStr(LightYTrack.Position * 0.5);
   LightZTrack.Position := convertRealForTracker(Info.LightZ);
   LightZLabel.Caption := FloatToStr(LightZTrack.Position * 0.5);
  end;

end;


{ ---------------------------------------------------------------------------

  Methodenname:         fillInfo
  Parameter
        Typ:            var Textfile
        Name:           cfg
        Bedeutung:      Zeiger auf eine geöffnete CFG-Datei

  Methodenbeschreibung:
  Liest den Inhalt einer CFG-Datei aus und füllt den Inhalt der Info-Struktur
  mit den Daten aus.

  --------------------------------------------------------------------------- }

procedure TMDIMain.fillInfo(var cfg : TextFile);

// Variablen mit einem Wert vorinitialisieren
const bmp    : boolean = false;         // Die Variablen werden auf true
      aa     : boolean = false;         // gesetzt, wenn deren gleichnamige
      files  : boolean = false;         // Option in der CFG-Datei gefunden wurde
      width  : boolean = false;
      height : boolean = false;
      pov    : boolean = false;
      qpov   : boolean = false;
      camx   : boolean = false;
      camy   : boolean = false;
      camz   : boolean = false;
      lightx : boolean = false;
      lighty : boolean = false;
      lightz : boolean = false;

var   s     : string;                 // Zwischenspeicher für ein String
      key   : string;                 // Speichert den Bezeichner einer Option
      value : string;                 // Speichert den Wert einer Option
      i, j  : integer;                // Laufvariablen

begin

 // Durchlaufe die CFG-Datei bis zum Ende
 while not eof(cfg) do
  begin

   ReadLn(cfg, s);      // Speicher eine Zeile in die Variable s
   trim(s);             // Schneide Leerzeichen am Anfang und am Ende ab
   j := Pos('//', s);   // Suche, ob die Zeile ein Kommentar enthält
   i := Pos('=', s);    // Suche, ob die Zeile eine Option enthält

   // Ist die Zeile eine Leerzeile oder eine Kommentarzeile, dann überspringe diese
   if (j = 1) or (length(s) = 0) or ((j > 0) and (i > j)) then
    continue;

   key := trim(Copy(s, 1, i-1));        // Zuweisung des Bezeichners der Option

   if (j <> 0) then
    value := trim(Copy(s, i+1, j-1-i))  // Zuweisung des Wertes und abschneiden
                                        // eines Kommentars
   else
```

```
 value := trim(Copy(s, i+1, length(s)));     // Zuweisung des Wertes

// Ist es der Bezeichner Image?
if CompareText(key, 'Image') = 0 then
 begin
  bmp := true;       // Option gefunden

  // Wenn der Wert bmp ist, dann setze die Variable in der Info-Struktur
  if CompareText(value, 'bmp') = 0 then
   Info.Bitmap := True

  // Wenn der Wert targa ist, dann setze nicht die Variable
  else if CompareText(value, 'targa') = 0 then
   Info.Bitmap := False

  // Falls nichts zutrifft, dann liegt kein gültiger Wert vor
  else
   bmp := false;
 end

// Ist es der Bezeichner Height?
else if CompareText(key, 'Height') = 0 then
 begin
  try
   Info.Height := StrToInt(value);   // Wandel den String in eine Zahl um
   height := true;                   // Option wurde gefunden
  except
   on EConvertError do ;             // Bei einem Konvertierungsfehler, mache nichts
  end;
 end

// Ist es der Bezeichner Width?
else if CompareText(key, 'Width') = 0 then
 begin
  try
   Info.Width := StrToInt(value);    // Wandel den String in eine Zahl um
   width := true;                    // Option wurde gefunden
  except
   on EConvertError do ;             // Bei einem Konvertierungsfehler, mache nichts
  end;
 end

// Ist es der Bezeichner AA?
else if CompareText(key, 'AA') = 0 then
 begin
  aa := true;        // Option gefunden

  // Wenn der Wert on ist, dann setze die Variable in der Info-Struktur
  if CompareText(value, 'on') = 0 then
   Info.AA := True

  // Wenn der Wert off ist, dann setze nicht die Variable
  else if CompareText(value, 'off') = 0 then
   Info.AA := False

  // Falls nichts zutrifft, dann liegt ein ungültiger Wert vor
  else
   aa := false;
 end

// Ist es der Bezeichner POV?
else if CompareText(key, 'POV') = 0 then
 begin

  // Ist es keine leere Zeichenkette?
  if length(value) <> 0 then
   begin
    Info.PathPOV := value;   // Übernehme den String
    pov := true;             // Option gefunden
   end;
 end

// Ist es der Bezeichner QPOV?
else if CompareText(key, 'QPOV') = 0 then
 begin

  // Ist es keine leere Zeichenkette?
  if length(value) <> 0 then
```

```
    begin
     Info.PathQPOV := value;  // Übernehme den String
     qpov := true;            // Option gefunden
    end;
 end

// Ist es der Bezeichner Files?
else if CompareText(key, 'Files') = 0 then
 begin
  files := true;     // Option gefunden

  // Wenn der Wert on ist, dann setze die Variable in der Info-Struktur
  if CompareText(value, 'on') = 0 then
   Info.Files := True

  // Wenn der Wert off ist, dann setze nicht die Variable
  else if CompareText(value, 'off') = 0 then
   Info.Files := False

  // Falls nichts zutrifft, dann liegt ein ungültiger Wert vor
  else
   files := false;
 end

// Ist es der Bezeichner CamX?
else if CompareText(key, 'CamX') = 0 then
 begin
  try
   Info.CamX := StrToFloat(value);   // Wandel den String in eine Fließkommazahl um
   camx := true;                     // Option gefunden
  except
   on EConvertError do ;             // Bei einem Konvertierungsfehler, mache nichts
  end;
 end

// Ist es der Bezeichner CamY?
else if CompareText(key, 'CamY') = 0 then
 begin
  try
   Info.CamY := StrToFloat(value);   // Wandel den String in eine Fließkommazahl um
   camy := true;                     // Option gefunden
  except
   on EConvertError do ;             // Bei einem Konvertierungsfehler, mache nichts
  end;
 end

// Ist es der Bezeichner CamZ?
else if CompareText(key, 'CamZ') = 0 then
 begin
  try
   Info.CamZ := StrToFloat(value);   // Wandel den String in eine Fließkommazahl um
   camz := true;                     // Option gefunden
  except
   on EConvertError do ;             // Bei einem Konvertierungsfehler, mache nichts
  end;
 end

// Ist es der Bezeichner LightX?
else if CompareText(key, 'LightX') = 0 then
 begin
  try
   Info.LightX := StrToFloat(value); // Wandel den String in eine Fließkommazahl um
   lightx := true;                   // Option gefunden
  except
   on EConvertError do ;             // Bei einem Konvertierungsfehler, mache nichts
  end;
 end

// Ist es der Bezeichner LightY?
else if CompareText(key, 'LightY') = 0 then
 begin
  try
   Info.LightY := StrToFloat(value); // Wandel den String in eine Fließkommazahl um
   lighty := true;                   // Option gefunden
  except
   on EConvertError do ;             // Bei einem Konvertierungsfehler, mache nichts
   end;
 end
```

```
   // Ist es der Bezeichner LightZ?
   else if CompareText(key, 'LightZ') = 0 then
    begin
     try
      Info.LightZ := StrToFloat(value); // Wandel den String in eine Fließkommazahl um
      lightz := true;                   // Option gefunden
     except
      on EConvertError do ;             // Bei einem Konvertierungsfehler, mache nichts
      end;
    end;

 end;          // Ende der While-Schleife

// Prüfe ob alle Optionen gefunden, bzw. einen gültigen Wert enthalten.
// Ist es nicht der Fall, dann initalisiere diese mit Standardwerten und gib
// eine Meldung an den Benutzer aus.
if not bmp then
 begin
  Info.Bitmap := True;
  MessageDlg('Für den Schlüssel "Image" wurde kein gültiger Wert gefunden!'
             +' Es wird der Standwert "bmp" verwendet!', mtWarning, [mbOK], 0);
 end
else if not width then
 begin
  Info.Width := 640;
  MessageDlg('Für den Schlüssel "Width" wurde kein gültiger Wert gefunden! '
              +'Es wird der Standwert "640" verwendet!', mtWarning, [mbOK], 0);
 end
else if not height then
 begin
  Info.Height := 480;
  MessageDlg('Für den Schlüssel "Height" wurde kein gültiger Wert gefunden!'
             +' Es wird der Standwert "480" verwendet!', mtWarning, [mbOK], 0);
 end
else if not files then
 begin
  Info.Files := false;
  MessageDlg('Für den Schlüssel "Files" wurde kein gültiger Wert gefunden!'
             +' Es wird der Standwert "false" verwendet!', mtWarning, [mbOK], 0);
 end
else if not pov then
 begin
  Info.PathPOV := 'POV=c:\programme\pov-ray for windows v3.5\bin\';
  MessageDlg('Für den Schlüssel "POV" wurde kein gültiger Wert gefunden!'
             +' Es wird ein Standwert verwendet!', mtWarning, [mbOK], 0);
 end
else if not qpov then
 begin
  Info.PathQPOV := 'QPOV=c:\programme\pov-ray for windows v3.5\guiext\quietpov\';
  MessageDlg('Für den Schlüssel "QPOV" wurde kein gültiger Wert gefunden!'
             +' Es wird ein Standwert verwendet!', mtWarning, [mbOK], 0);
 end
else if not aa then
 begin
  Info.Bitmap := false;
  MessageDlg('Für den Schlüssel "AA" wurde kein gültiger Wert gefunden!'
             +' Es wird der Standwert "off" verwendet!', mtWarning, [mbOK], 0);
 end
else if not camx then
 begin
  Info.CamX := 2.0;
  MessageDlg('Für den Schlüssel "CamX" wurde kein gültiger Wert gefunden!'
             +' Es wird der Standwert "2.0" verwendet!', mtWarning, [mbOK], 0);
 end
else if not camy then
 begin
  Info.CamY := 2.0;
  MessageDlg('Für den Schlüssel "Camy" wurde kein gültiger Wert gefunden!'
             +' Es wird der Standwert "2.0" verwendet!', mtWarning, [mbOK], 0);
 end
else if not camz then
 begin
  Info.CamZ := 2.0;
  MessageDlg('Für den Schlüssel "CamZ" wurde kein gültiger Wert gefunden!'
             +' Es wird der Standwert "2.0" verwendet!', mtWarning, [mbOK], 0);
 end
else if not lightx then
```

```
  begin
   Info.LightX := 1.0;
   MessageDlg('Für den Schlüssel "LightX" wurde kein gültiger Wert gefunden!'
              +' Es wird der Standwert "1.0" verwendet!', mtWarning, [mbOK], 0);
  end
 else if not lighty then
  begin
   Info.LightY := 1.0;
   MessageDlg('Für den Schlüssel "LightY" wurde kein gültiger Wert gefunden!'
              +' Es wird der Standwert "1.0" verwendet!', mtWarning, [mbOK], 0);
  end
 else if not lightz then
  begin
   Info.LightZ := 1.0;
   MessageDlg('Für den Schlüssel "LightZ" wurde kein gültiger Wert gefunden!'
              +' Es wird der Standwert "1.0" verwendet!', mtWarning, [mbOK], 0);
  end;

end;


{ ---------------------------------------------------------------------------

  Methodenname:         defaultCFG

  Methodenbeschreibung:
  Schreibt eine CFG-Datei mit Standardwerten.

  --------------------------------------------------------------------------- }

procedure TMDIMain.defaultCFG;

var cfg : TextFile;             // File-Handle für die CFG-Datei

begin

 // Erzeugen der CFG-Datei
 AssignFile(cfg, 'LPROCESS.CFG');       // File-Handle den Namen zuweisen
 {$i+} Rewrite(cfg); {$i+}              // Öffnen der Datei zum schreiben und I/O-Prüfung
                                        // des Compilers temporär ausschalten
 if IOResult <> 0 then                  // Falls die Datei nicht geöffnet werden konnte,
  begin                                 // dann gib eine Fehlermeldung an den Benutzer
                                        // aus und Beende die Funktion.
   MessageDlg('Das Programm kann nicht gestartet werden, weil keine CFG-Datei erstellt'
              +' werden kann!', mtError, [mbOk], 0);
   Close;
  end;

 // Schreibt die Standardwerte in die neue CFG-Datei
 WriteLn(cfg, 'Image=bmp');
 WriteLn(cfg, 'Height=480');
 WriteLn(cfg, 'Width=640');
 WriteLn(cfg, 'AA=off');
 WriteLn(cfg, 'POV=c:\programme\pov-ray for windows v3.5\bin\');
 WriteLn(cfg, 'QPOV=c:\programme\pov-ray for windows v3.5\guiext\quietpov\');
 WriteLn(cfg, 'Files=off');
 WriteLn(cfg, 'CamX=2.0');
 WriteLn(cfg, 'CamY=2.0');
 WriteLn(cfg, 'CamZ=2.0');
 WriteLn(cfg, 'LightX=1.0');
 WriteLn(cfg, 'LightY=1.0');
 WriteLn(cfg, 'LightZ=1.0');

 // Initialisiert die Info-Struktur ebenfalls mit den Standardwerten
 Info.Bitmap := true;
 Info.AA := false;
 Info.Files := false;
 Info.Width := 640;
 Info.Height := 480;
 Info.CamX := 2.0;
 Info.CamY := 2.0;
 Info.CamZ := 2.0;
 Info.LightX := 1.0;
 Info.LightY := 1.0;
 Info.LightZ := 1.0;
 Info.PathPOV := 'POV=c:\programme\pov-ray for windows v3.5\bin\';
 Info.PathQPOV := 'QPOV=c:\programme\pov-ray for windows v3.5\guiext\quietpov\';
```

```
 CloseFile(cfg);        // Schließen der CFG-Datei

end;


{ ---------------------------------------------------------------------------

  Methodenname:         checkLSLine
  Rückgabewert
        Typ:            string
        Bedeutung:      Gibt einen String mit einer Fehlermeldung zurück oder
                        eine leere Zeichenkette
  Parameter
        Typ:            var string
        Name:           line
        Bedeutung:      Zeiger auf eine Zeile

  Methodenbeschreibung:
  Prüft eine Zeile aus einer LS-Datei, ob diese eine gültige Produktion ist.

  --------------------------------------------------------------------------- }

function TMDIMain.checkLSLine(var line: string) : string;

var s           : string;       // Zwischenspeicher für ein String
    production  : string;       // Enthält die fertige Produktion
    lcon        : string;       // Linker Kontext der Produktion
    pred        : string;       // Produktionskopf der Produktion
    rcon        : string;       // Rechter Kontext der Produktion
    cond        : string;       // Bedingungsteil der Produktion
    succ        : string;       // Produktionskörper der Produktion
    i           : integer;      // Speichert die Position des linken Kontext Zeichens
    j           : integer;      // Speichert die Position des Anfangs des Produktionskörpers
    k           : integer;      // Speichert die Position des rechten Kontext Zeichens
    l           : integer;      // Speichert die Position des Bedingungszeichens

begin

 result := '';          // Rückgabewert mit einem Leerstring initialisieren

 i := Pos('<', line);   // Zuweisung der Position des linken Kontext Zeichens
 k := Pos('>', line);   // Zuweisung der Position des rechten Kontext Zeichens
 l := Pos(':', line);   // Zuweisung der Position des Bedingungszeichens

 // Falls das linke Kontext Zeichen ganz links außen steht,
 // dann ist die Produktion nicht richtig formuliert.
 if i = 1 then
  begin
   result := 'Linker Kontext fehlt!';
   Exit;
  end;

 s := trim(Copy(line, 1, i-1)); // Der linke Kontext wird aus dem String herauskopiert

 // Ist das Wildcardzeichen enthalten und weitere Zeichen?
 if (Pos('*', s) > 0) and (length(s) > 1) then
  begin         // Gebe Fehlermeldung zurück und Beende die Verarbeitung
   result := 'Der linke Kontext enthält neben dem *-Zeichen, noch weitere Zeichen!';
   exit;
  end;

 // Ist das Wildcardzeichen nicht vorhanden, aber runde Klammern?
 if (Pos('*', s) = 0) and ((Pos('(', s) > 0) or (Pos(')', s) > 0)) then
  begin         // Gebe Fehlermeldung zurück und Beende die Verarbeitung
   result := 'Im Kontext sind keine Klammern erlaubt!';
   Exit;
  end;

 // Ist das linke Kontext Zeichen in echt ein Teil des Bedingungsteils?
 if (l > 0) and (i > l) then
  begin
   s := '*';
   i := 0;
  end;

 // Ist kein linker Kontext vorhanden?
 if ((Pos('*', s) > 0) and (length(s) = 1)) or (length(s) = 0) then
  lcon := ''
```

```
else   // ansonsten baue den linken Kontext zusammen
 lcon := s+' < ';

j := Pos('->', line);  // Zuweisung der Position des Anfangs des Produktionskörpers
s := trim(Copy(line, j+2, length(line)));      // Kopiere den Produktionskörper heraus

// Enthält die Zeichenkette etwas?
if length(s) = 0 then
 begin         // Gebe Fehlermeldung zurück und Beende die Verarbeitung
  result := 'Es existiert kein rechter Produktionsteil!';
  Exit;
 end;
succ := ' -> '+s;      // Baue den Produktionskörper zusammen

// Existiert kein Bedingungsteil und kein rechter Kontext?
if (k = (j+1)) and (l = 0) then
 begin
  rcon := '';
  cond := '';
  pred := trim(Copy(line, i+1, j-1-i));        // Kopiere den Produktionskopf heraus
 end

// Existiert ein rechter Kontext, aber kein Bedingungsteil?
else if (k <> (j+1)) and (l = 0) then
 begin
  s := trim(Copy(line, k+1, j-1-k));   // Kopiere den rechten Kontext heraus

  // Existiert ein Inhalt für den rechten Kontext?
  if length(s) = 0 then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Rechter Kontext fehlt!';
    exit;
   end;

  // Sind weitere Zeichen außer dem Wildcardzeichen im rechten Kontext enthalten?
  if (Pos('*', s) > 0) and (length(s) > 1) then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Der linke Kontext enthält neben dem *-Zeichen, noch weitere Zeichen!';
    exit;
   end;

  // Sind Klammern im rechten Kontext enthalten?
  if (Pos('*', s) = 0) and ((Pos('(', s) > 0) or (Pos(')', s) > 0)) then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Im Kontext sind keine Klammern erlaubt!';
    Exit;
   end;

  // Enthält der rechte Kontext keinen Ausdruck?
  if ((Pos('*', s) > 0) and (length(s) = 1)) or (length(s) = 0) then
   rcon := ''
  else         // Ansonsten baue einen rechten Kontext zusammen
   rcon := ' > '+s;
  cond := '';
  pred := trim(Copy(line, i+1, k-1-i));        // Kopiere den Produktionskopf heraus
 end

// Existiert ein Bedingungsteil, aber kein rechter Kontext?
else if (k > l) and (l > 0) then
 begin
  s := trim(Copy(line, l+1, j-1-l));           // Kopiere den Bedingungsteil heraus
  pred := trim(Copy(line, i+1, l-1-i));        // Kopiere den Produktionskopf heraus
  rcon := '';

  // Existiert ein Inhalt für den Bedingungsteil?
  if length(s) = 0 then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Bedingung fehlt!';
    exit;
   end;

  // Sind weitere Zeichen außer dem Wildcardzeichen im Bedingungsteil enthalten?
  if (Pos('*', s) > 0) and (length(s) > 1) then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Die Bedingung enthält neben dem *-Zeichen, noch weitere Zeichen!';
    exit;
   end;
```

```
  // Sind Klammern im Bedingungsteil enthalten?
  if (Pos('*', s) = 0) and ((Pos('(', s) > 0) or (Pos(')', s) > 0)) then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'In der Bedingung sind keine Klammern erlaubt!';
    Exit;
   end;

  // Enthält der Bedingungsteil keinen Ausdruck?
  if ((Pos('*', s) > 0) and (length(s) = 1)) or (length(s) = 0) then
   cond := ''
  else         // Ansonsten baue einen Bedingungsteil zusammen
   cond := ' : '+s;
 end

// Es existert sowohl ein Bedingungsteil, als auch ein rechter Kontext!
else
 begin
  pred := trim(Copy(line, i+1, k-1-i));        // Kopiere den Produktionskopf heraus
  s := trim(Copy(line, l+1, j-1-l));           // Kopiere den Bedingungsteil heraus

  // Existiert ein Inhalt für den Bedingungsteil?
  if length(s) = 0 then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Bedingung fehlt!';
    exit;
   end;

  // Sind weitere Zeichen außer dem Wildcardzeichen im Bedingungsteil enthalten?
  if (Pos('*', s) > 0) and (length(s) > 1) then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Die Bedingung enthält neben dem *-Zeichen, noch weitere Zeichen!';
    exit;
   end;

  // Sind Klammern im Bedingungsteil enthalten?
  if (Pos('*', s) = 0) and ((Pos('(', s) > 0) or (Pos(')', s) > 0)) then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'In der Bedingung sind keine Klammern erlaubt!';
    Exit;
   end;

  // Enthält der Bedingungsteil keinen Ausdruck?
  if ((Pos('*', s) > 0) and (length(s) = 1)) or (length(s) = 0) then
   cond := ''
  else         // Ansonsten baue einen Bedingungsteil zusammen
   cond := ' : '+s;

  s := trim(Copy(line, k+1, l-1-k));           // Kopiere den rechten Kontext heraus

  // Existiert ein Inhalt für den rechten Kontext?
  if length(s) = 0 then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Rechter Kontext fehlt!';
    exit;
   end;

  // Sind weitere Zeichen außer dem Wildcardzeichen im rechten Kontext enthalten?
  if (Pos('*', s) > 0) and (length(s) > 1) then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Der rechte Kontext enthält neben dem *-Zeichen, noch weitere Zeichen!';
    exit;
   end;

  // Sind Klammern im rechten Kontext enthalten?
  if (Pos('*', s) = 0) and ((Pos('(', s) > 0) or (Pos(')', s) > 0)) then
   begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
    result := 'Im Kontext sind keine Klammern erlaubt!';
    Exit;
   end;

  // Enthält der rechte Kontext keinen Ausdruck?
  if ((Pos('*', s) > 0) and (length(s) = 1)) or (length(s) = 0) then
   rcon := ''
  else         // Ansonsten baue einen rechten Kontext zusammen
   rcon := ' > '+s;
 end;

// Existiert ein Inhalt für den Produktionskopf
```

```
 if length(pred) = 0 then
  begin       // Gebe Fehlermeldung zurück und Beende die Verarbeitung
   result := 'Es existert kein Produktionskopf!';
   exit;
  end;
 result := testingProductionHead(pred); // Lass den Produktionskopf prüfen auf
                                        // seine Gültigkeit
 if result <> '' then   // Ist der Produktionskopf ungültig?
  exit;
 if cond <> '' then     // Existiert ein Bedingungsteil?
  begin
   result := testingCondition(Copy(cond, 4, length(cond)));     // Prüfen des Bedingungsteils
                                                                // auf seine Gültigkeit
   if result <> '' then // Ist der Bedingungsteil ungültig?
    exit;
  end;
 production := lcon + pred + rcon + cond + succ;        // Zusammensetzen der Produktion
 line := production;                                    // Produktion zurückgeben

end;


{ ---------------------------------------------------------------------------

  Methodenname:         CreateMDIChild
  Parameter
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           Name
        Bedeutung:      Enthälten den Titel für das neue Kindfenster

  Methodenbeschreibung:
  Erzeugt ein neues Kindfenster mit dem übergebenen Namen.

  --------------------------------------------------------------------------- }

procedure TMDIMain.CreateMDIChild(const Name: string);

var Child :  TMDIChildMain;     // Zu erzeugende Instanz des Kindfensters

begin
 Child := TMDIChildMain.Create(Application);    // Erzeugen des Kindfensters
 Child.Caption := Name;                         // Titel dem Kindfenster übergeben
 setOptionsForChild(Child);                     // Einstellungen ans Kindfenster übergeben
end;


{ ---------------------------------------------------------------------------

  Methodenname:         HelpAboutExecute
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Erzeugt das Info-Fenster und zeigt es an.

  --------------------------------------------------------------------------- }

procedure TMDIMain.HelpAboutExecute(Sender: TObject);

var about : TAboutForm; // Zu erzeugende Instanz des Info-Fensters

begin
 about := TAboutForm.Create(Application);       // Erzeuge Instanz des Info-Fensters
 about.ShowModal;                               // Fenster modal anzeigen
 about.Release;                                 // Speicher freigeben, wenns geschlossen wird
end;


{ ---------------------------------------------------------------------------

  Methodenname:         FileNewExecute
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
```

```
  Methodenbeschreibung:
  Erzeugt ein neues und leeres Kindfenster.

  ------------------------------------------------------------------------ }

procedure TMDIMain.FileNewExecute(Sender: TObject);

begin
 CreateMDIChild('Unbenannt'+IntToStr(MDICHildCount+1)); // Erzeuge neues Kindfenster
end;


{ ---------------------------------------------------------------------------

  Methodenname:         OptionsCFGExecute
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Erzeugt das Einstellungs-Formular und zeigt es an.

  ------------------------------------------------------------------------ }

procedure TMDIMain.OptionsCFGExecute(Sender: TObject);

var CFG : TCFGForm;     // Zu erzeugende Instanz des Einstellungs-Formulars

begin
 CFG := TCFGForm.Create(Application);   // Erzeuge Instanz
 CFG.ShowModal;                        // Zeige das Formular modal an
end;


{ ---------------------------------------------------------------------------

  Methodenname:         FileOpen1Accept
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Öffnet eine Datei und erzeugt ein Kindfenster mit dem Inhalt der Datei.

  ------------------------------------------------------------------------ }

procedure TMDIMain.FileOpen1Accept(Sender: TObject);

begin
 openLS(FileOpen1.Dialog.FileName);     // Erzeuge Kindfenster mit dem Inhalt der Datei
end;


{ ---------------------------------------------------------------------------

  Methodenname:         FormCreate
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Methode wird beim erzeugen des Formulars aufgerufen. Es läd die Info-Struktur
  mit den notwendigen Informationen.

  ------------------------------------------------------------------------ }

procedure TMDIMain.FormCreate(Sender: TObject);

var cfg : TextFile;     // File-Handle für die CFG-Datei

begin
 DecimalSeparator := '.';       // Setzt fest, dass der Punkt den Nachkommateil symbolisiert

 AssignFile(cfg, 'LPROCESS.CFG');       // Weise den File-Handle den Namen der Datei zu
 {$i-} Reset(cfg); {$i+}                // Öffne Datei zum lesen und deaktiviere temporär
```

```
                                        // die I/O-Prüfung
 if IOResult <> 0 then                  // Konnte die Datei nicht geöffnet werden?
  begin
   defaultCFG;                          // Dann erzeuge eine Standard CFG-Datei
   Exit;                                // und Verlasse die Methode
  end;

 fillInfo(cfg);         // Füll den Inhalt der Info-Struktur mit den Daten der CFG-Datei
 CloseFile(cfg);        // Schließe die CFG-Datei
end;


{ ---------------------------------------------------------------------------

  Methodenname:         FormClose
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var TCloseAction
        Name:           Action
        Bedeutung:      Gibt an, wie das Fenster geschlossen werden ssoll

  Methodenbeschreibung:
  Methode wird beim Beenden des Formulars aufgerufen.

  --------------------------------------------------------------------------- }

procedure TMDIMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 WinExec('lprocess -e', SW_HIDE);       // Beende eine laufende POV-Ray Instanz
end;


{ ---------------------------------------------------------------------------

  Methodenname:         FormCloseQuery
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var Boolean
        Name:           CanClose
        Bedeutung:      Gibt an, ob das Formular geschlossen werden soll.

  Methodenbeschreibung:
  Methode wird vor dem Schließen des Formulars aufgerufen.

  --------------------------------------------------------------------------- }

procedure TMDIMain.FormCloseQuery(Sender: TObject; var CanClose: Boolean);

var cfg : TextFile;     // File-Handle für die CFG-Datei

begin
 CanClose := true;      // Applikaton kann geschlossen werden

 AssignFile(cfg, 'LPROCESS.CFG');       // Weise dem File-Handle den Namen der Datei zu
 {$i-} Rewrite(cfg); {$i+}              // Öffne die Datei zum lesen
 if IOResult <> 0 then                  // Konnte Datei nicht geöffnet werden?
  begin

   // Soll die Applikation wirklich beendet werden?
   if MessageDlg('Die Einstellungen für die CFG-Datei können nicht gespeichert werden!'+
                 #13+'Soll die Anwendung trotzdem geschlossen werden?', mtError,
                 [mbYes, mbNo], 0) = mrNo then
    CanClose := false;
   Exit;
  end;

 // Speicher die Daten aus der Info-Struktur in die CFG-Datei
 if Info.Bitmap = true then
  WriteLn(cfg, 'Image=bmp')
 else
  WriteLn(cfg, 'Image=Targa');
 if Info.AA = true then
  WriteLn(cfg, 'AA=on')
 else
```

```
  WriteLn(cfg, 'AA=off');
 if Info.Files = true then
  WriteLn(cfg, 'Files=on')
 else
  WriteLn(cfg, 'Files=off');
 WriteLn(cfg, 'Width='+IntToStr(Info.Width));
 WriteLn(cfg, 'Height='+IntToStr(Info.Height));
 WriteLn(cfg, 'POV='+Info.PathPOV);
 WriteLn(cfg, 'QPOV='+Info.PathQPOV);
 WriteLn(cfg, 'CamX='+FloatToStr(Info.CamX));
 WriteLn(cfg, 'CamY='+FloatToStr(Info.CamY));
 WriteLn(cfg, 'CamZ='+FloatToStr(Info.CamZ));
 WriteLn(cfg, 'LightX='+FloatToStr(Info.LightX));
 WriteLn(cfg, 'LightY='+FloatToStr(Info.LightY));
 WriteLn(cfg, 'LightZ='+FloatToStr(Info.LightZ));
 CloseFile(cfg);                        // Schließe die CFG-Datei
end;


{ ---------------------------------------------------------------------------

  Methodenname:         ImportLSExecute
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Konvertiert den Inhalt einer LS-Datei um und zeigt den Inhalt in einem neuen
  Kindfenster an.

  --------------------------------------------------------------------------- }

procedure TMDIMain.ImportLSExecute(Sender: TObject);

var s, t : string;      // Temporäre Zwischenspeicherung von Strings
    i    : integer;     // Zwischenspeicher

begin

 // Öffne eine Standard-Dialogbox zum öffnen einer Datei.
 // Ist Ausführung erfolgreich gewesen (gültige Auswahl einer Datei?)
 if OpenDialog1.Execute then
  begin

   // Soll der Inhalt in eine neue Datei gespeichert werden?
   if MessageDlg('Soll eine neu Datei angelegt werden, in der die Daten in Version 5'
               +' vorliegen?'+#13+'Andernfalls werden die Daten in der alten Datei'
               +' überschrieben!', mtInformation, [mbYes, mbNo], 0) = mrYes then
    begin

     // Rufe eine InputeBox auf, damit der Anwender den Dateinamen eingeben kann
     s := InputBox('Dateiname eingeben!', 'Bitte geben Sie ein Dateinamen an, in der'
                  +' die neuen Daten gespeichert werden soll!', '');

     // Wenn kein Dateiname angegeben wurde, dann Beende die Methode
     if s = '' then
      Exit;
     i := length(s);    // Hole die Länge der Zeichenkette

     // Prüfte die Dateinamenerweiterung
     if not((s[i-2] = '.') and (s[i-1] = 'l') and (s[i] = 's')) then
      s := s+'.ls';

     PicName := GetCurrentDir+'\'+s;    // Definiere den vollständigen Pfad
     FileOpen1.Dialog.FileName := s;    // Zuweisung des neuen Dateinamens
     t := ExtractFileName(OpenDialog1.FileName);        // Hole Dateinamen der zu
                                                        // konvertierenden Datei

     // Definiere die Zeichenkette für den Kommandointerpreter
     JvCreateProcess1.CommandLine := 'lprocess -c '+Copy(t, 1, length(t)-3)+' '+s;
    end
   else
    begin
     FileOpen1.Dialog.FileName := OpenDialog1.FileName;
     t := ExtractFileName(OpenDialog1.FileName);        // Hole Dateinamen der zu
                                                        // konvertierenden Datei
```

```
     // Definiere die Zeichenkette für den Kommandointerpreter
     JvCreateProcess1.CommandLine := 'lprocess -c '+Copy(t, 1, length(t)-3);
     PicName := OpenDialog1.FileName;
    end;
   JvCreateProcess1.Tag := 2;           // Signalisiere, dass eine Umwandlung
                                        // stattfinden wird
   ConsoleFom.ListBox1.Clear;           // Lösche den Inhalt der Konsolenausgabe
   JvCreateProcess1.Run;                // Führe die Anweisung aus
  end;
end;


{ ---------------------------------------------------------------------------

  Methodenname:         JvCreateProcess1Read
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            string (konstante Zeichenkette, Geschwindigkeitsoptimierung)
        Name:           S
        Bedeutung:      Enthält die Konsolenausgabe

  Methodenbeschreibung:
  Methode wird ausgeführt, wenn Zeichenketten von der Standardeingabe
  an die Komponente übergeben werden.

  --------------------------------------------------------------------------- }

procedure TMDIMain.JvCreateProcess1Read(Sender: TObject; const S: String);

begin
 // Schreibe die Konsolenausgabe in die Listbox des Formulars für die Konsolenausgabe
 ConsoleFom.ListBox1.Items.Add(s);
end;


{ ---------------------------------------------------------------------------

  Methodenname:         JvCreateProcess1Terminate
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            Cardinal
        Name:           ExitCode
        Bedeutung:      Enthält den Exitcode der ausgeführten Anwendung

  Methodenbeschreibung:
  Führt eine bestimmte Anwendung aus, die vorher in dem Attribut CommandLine
  definiert wurde.

  --------------------------------------------------------------------------- }

procedure TMDIMain.JvCreateProcess1Terminate(Sender: TObject;
  ExitCode: Cardinal);

var showpic : TShowPicForm;     // Zu erzeugende Instanz eines Formulars zum Anzeigen
                                // eines Bildes

begin

 // Ist die Anwendung nicht erfolgreich beendet worde und ist die Ausführung
 // eine Bilderzeugung?
 if (ExitCode <> 0) and (JvCreateProcess1.Tag = 1) then
  begin         // Gib eine Fehlermeldung aus und zeige dem Anwender die Konsolenausgabe
   MessageDlg('Bild konnte nicht erzeugt werden, weil ein Fehler vor lag!'+#13
              +'Die Ausgabe der Konsole wird eingeblendet!', mtError, [mbOk], 0);
   ConsoleFom.ShowModal;
   ConsoleFom.Release;
  end

 // Ist die Ausführung eine Bilderzeugung und ist diese erfolgreich?
 else if JvCreateProcess1.Tag = 1 then
  begin
   showpic := TShowPicForm.Create(Application);         // Erzeuge Instanz
   showpic.Image1.Picture.LoadFromFile(PicName);        // Lade das Bild
   showpic.Height := showpic.Image1.Picture.Height;     // Anpassen der Größe des Formulars
```

```
   showpic.Width := showpic.Image1.Picture.Width;       // an die Größe des Bildes
   showpic.ShowModal;                                   // Anzeige des Formulars modal
  end

 // Ist die Anwendung nicht erfolgreich beendet worde und ist die Ausführung
 // eine Konvertierung?
 else if (ExitCode <> 0) and (JvCreateProcess1.Tag = 2) then
  begin         // Gib eine Fehlermeldung aus und zeige dem Anwender die Konsolenausgabe
   MessageDlg('Datei konnte nicht importiert werden!'+#13+'Die Ausgabe der Konsole'
              +' wird eingeblendet!', mtError, [mbOk], 0);
   ConsoleFom.ShowModal;
   ConsoleFom.Release;
  end

 // Ist die Ausführung eine Konvertierung und ist diese erfolgreich?
 else if JvCreateProcess1.Tag = 2 then
  openLS(PicName);              // Öffne die Datei und zeigen den Inhalt in einem
                                // neuen Kindfenster
end;

end.
```

Dateiname: ShowPicSource.pas

```
{ ---------------------------------------------------------------------------

  Autor:                Jan Derer
  Datum:                05. 06. 04
  Kontakt:              Darkhabits@aol.com
  Programmname:         VisualL
  Version:              1.0

  Klassenname:          TShowPicForm
  Version:              1.0
  Kurzbeschreibung:
  Diese Klasse repräsentiert das Fenster, das zum Anzeigen der Grafik benutzt
  wird.

  --------------------------------------------------------------------------- }

unit ShowPicSource;

{ ---------------------------------------------------------------------------
                BESCHREIBUNG DER SCHNITTSTELLE DER UNIT
  --------------------------------------------------------------------------- }

interface

{ ---------------------------------------------------------------------------
                Liste alle öffentlich eingebundenen Units
  --------------------------------------------------------------------------- }

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls;

{ ---------------------------------------------------------------------------
                    Deklaration eigener Datentypen
  --------------------------------------------------------------------------- }

type

{ ---------------------------------------------------------------------------
                   Klassenbeschreibung für TShowPicForm
  --------------------------------------------------------------------------- }

  TShowPicForm = class(TForm)

    // Auflistung aller eingebundenen Komponenten der Klasse
    Image1: TImage;

    // Auflistung aller Methoden für die Ereignisverarbeitung
    procedure Image1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
    { Private-Deklarationen }
  public
```

```
    { Public-Deklarationen }
  end;

{ ------------------------------------------------------------------------
                          Öffentliche globale Variablen
  ------------------------------------------------------------------------ }

var
  ShowPicForm: TShowPicForm;    // Die Instanz von TShowPicForm ist öffentlich für alle ande-
ren
                                // Units erreichbar

{ ------------------------------------------------------------------------
                          IMPLEMENTATIONSTEIL DER UNIT
  ------------------------------------------------------------------------ }

implementation

{ ------------------------------------------------------------------------
                               Compiler-Schalter
  ------------------------------------------------------------------------ }

{$R *.dfm}

{ ------------------------------------------------------------------------
                   Liste aller private eingebundenen Units
  ------------------------------------------------------------------------ }

uses GraphicEx;


{ ------------------------------------------------------------------------

  Methodenname:         Image1Click
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Schließt das Formular, wenn auf das Bild gelickt wird.

  ------------------------------------------------------------------------ }

procedure TShowPicForm.Image1Click(Sender: TObject);

begin
 Close;         // Schließe das Formular
end;


{ ------------------------------------------------------------------------

  Methodenname:         FormClose
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var TCloseAction
        Name:           Action
        Bedeutung:      Gibt an, wie das Fenster geschlossen werden ssoll

  Methodenbeschreibung:
  Methode wird beim Beenden des Formulars aufgerufen.

  ------------------------------------------------------------------------ }

procedure TShowPicForm.FormClose(Sender: TObject;
  var Action: TCloseAction);

begin
 Action := caFree;      // Speicher freigeben
end;

end.
```

Dateiname: TurtleComSource.pas

```
{ ---------------------------------------------------------------------------

  Autor:                Jan Derer
  Datum:                05. 06. 04
  Kontakt:              Darkhabits@aol.com
  Programmname:         VisualL
  Version:              1.0

  Klassenname:          TTurtleComForm
  Version:              1.0
  Kurzbeschreibung:
  Diese Klasse repräsentiert das Fenster, das zum Anzeigen der
  Turtle-Kommandos benutzt wird.

  --------------------------------------------------------------------------- }

unit TurtleComSource;

{ ---------------------------------------------------------------------------
                BESCHREIBUNG DER SCHNITTSTELLE DER UNIT
  --------------------------------------------------------------------------- }

interface

{ ---------------------------------------------------------------------------
                Liste alle öffentlich eingebundenen Units
  --------------------------------------------------------------------------- }

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, Buttons;

{ ---------------------------------------------------------------------------
                    Deklaration eigener Datentypen
  --------------------------------------------------------------------------- }

type

{ ---------------------------------------------------------------------------
                  Klassenbeschreibung für TTurtleComForm
  --------------------------------------------------------------------------- }

  TTurtleComForm = class(TForm)

    // Auflistung aller eingebundenen Komponenten der Klasse
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label3: TLabel;
    Label9: TLabel;
    Label11: TLabel;
    Label5: TLabel;
    Label7: TLabel;
    GroupBox2: TGroupBox;
    CloseBtn: TBitBtn;
    Label13: TLabel;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Label18: TLabel;
    Label19: TLabel;
    GroupBox3: TGroupBox;
    Label20: TLabel;
    Label21: TLabel;
    Label22: TLabel;
    Label23: TLabel;
    Label24: TLabel;
    Label25: TLabel;
    Label26: TLabel;
    Label27: TLabel;
    Label28: TLabel;
    Label29: TLabel;
    Label30: TLabel;
    Label31: TLabel;
    Label32: TLabel;
```

```
    Label33: TLabel;
    GroupBox4: TGroupBox;
    Label34: TLabel;
    Label35: TLabel;
    Label36: TLabel;
    Label37: TLabel;
    GroupBox5: TGroupBox;
    Label38: TLabel;
    Label39: TLabel;
    Label40: TLabel;
    Label41: TLabel;
    Label42: TLabel;
    Label43: TLabel;
    Label44: TLabel;
    Label45: TLabel;
    Label46: TLabel;
    Label2: TLabel;
    Label4: TLabel;
    Label6: TLabel;

    // Auflistung aller Methoden für die Ereignisverarbeitung
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure CloseBtnClick(Sender: TObject);
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

{ ---------------------------------------------------------------------------
                          Öffentliche globale Variablen
  --------------------------------------------------------------------------- }

var
  TurtleComForm: TTurtleComForm;        // Die Instanz von TTurtleComForm ist öffentlich für
alle anderen
                                        // Units erreichbar

{ ---------------------------------------------------------------------------
                          IMPLEMENTATIONSTEIL DER UNIT
  --------------------------------------------------------------------------- }

implementation

{ ---------------------------------------------------------------------------
                               Compiler-Schalter
  --------------------------------------------------------------------------- }

{$R *.dfm}

{ ---------------------------------------------------------------------------
                   Liste aller private eingebundenen Units
  --------------------------------------------------------------------------- }

uses MDIMainSource;


{ ---------------------------------------------------------------------------

  Methodenname:         FormClose
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft
        Typ:            var TCloseAction
        Name:           Action
        Bedeutung:      Gibt an, wie das Fenster geschlossen werden ssoll

  Methodenbeschreibung:
  Methode wird beim Beenden des Formulars aufgerufen.

  --------------------------------------------------------------------------- }

procedure TTurtleComForm.FormClose(Sender: TObject;
  var Action: TCloseAction);

begin
 TurtleActive := false;         // Signalisieren, dass es keine Instanz mehr gibt
```

```
 Action := caFree;              // Speicher freigeben
end;


{ ---------------------------------------------------------------------------

  Methodenname:         CloseBtnClick
  Parameter
        Typ:            TObject
        Name:           Sender
        Bedeutung:      Enthält das Objekt, das diese Methode aufruft

  Methodenbeschreibung:
  Schließt das Formular, wenn auf "Schließen" gelickt wird.

  --------------------------------------------------------------------------- }

procedure TTurtleComForm.CloseBtnClick(Sender: TObject);

begin
 Close;         // Schließe das Formular
end;

end.
```